intel

APPLICATION NOTE

AP-273

November 1986

Developing MCS®-96 Applications Using the SBE-96

DAVE SCHOEBEL
DSO APPLICATIONS

Order Number: 280249-001

INTRODUCTION

With the increasing demands of industrial and computer control applications, today's designers are looking for solutions whose performance extends beyond that of conventional 8-bit architectures. Traditionally, these control system architects must depend upon expensive and complex multi-chip microprocessors to achieve this high performance, but now a 16-bit single chip microcontroller can offer a much more cost-effective solution. Microcontrollers are microprocessors specially configured to monitor and control mechanisms and processes rather than manipulate data. They include CPU, program memory, data memory and a array of specialized peripherals on chip to produce a low component count solution. The MCS-96 family uses 120,000 transistors to implement a high performance 16-bit CPU, 8K bytes of program memory, 232 bytes of data memory and both analog and digital I/O features. Supporting this device are a suite of development tools hosted on both Intel development systems (Series III and IV) and industry standard hosts (IBM PC XT and PC AT).

This application note includes a brief description of the MCS-96 family of microcontrollers, its software development environment and hardware debugging centered around the iSBE-96 Single Board Emulator. Also included are helpful hints and programs to enable you to get the most from your investment dollars. The application note is partitioned into two sections. The first section introduces the MCS-96 architecture and development environment while the later section provides in-depth details of the iSBE-96 including its customization to your particular environment.

MCS®-96 MICROCONTROLLER OVERVIEW

Introduction to the MCS®-96 Architecture

The MCS-96 architecture consists of a 16-bit central processing unit (CPU) and a multitude of peripheral and I/O functions integrated into a single silicon component as shown in Figure 1. The CPU supports bit, byte and word operations. Double words (32-bits) are also supported in a subset of the instruction set. With a 12 MHz input crystal frequency, the MCS-96 microcontroller can perform a 16-bit addition in 1.0 microseconds (μs) and a 16 x 16 bit multiply or 32/16 bit divide in just 6.5 μs.

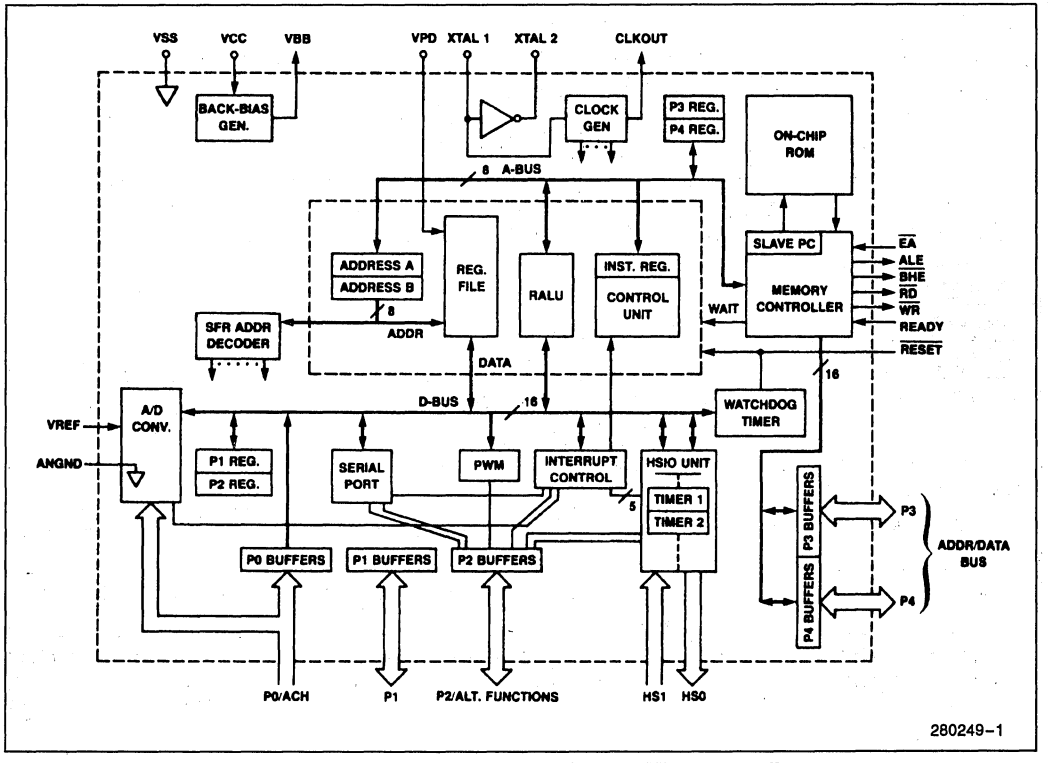

Figure 1. Block Diagram of the MCS®-96 Microcontroller

There are four high-speed trigger inputs that can record the times at which external events occur as often as every 2 μs (at 12 MHz crystal frequency). Up to six high-speed pulse generator outputs can trigger external events at pre-selected times. Additionally, the high-speed output unit can simultaneously perform timer functions. Up to four 16-bit software timers can be in operation simultaneously, in addition to the two 16-bit hardware timers. This makes the MCS-96 microcontroller particularly useful in process and control applications.

There is an optional on-chip analog to digital (A/D) converter which can convert up to four (in the 48-pin version) or eight (in the 68-pin version) analog input channels (10-bits resolution) in only 22 μs for the 8x9xBH parts or 42 μs for the 8x9x-90 parts.

Also provided on-chip is a full duplex serial port, dedicated baud rate generator, 16-bit watchdog timer, and a pulsewidth modulated output signal. Table 1 shows the features summary for the MCS-96 microcontroller. Table 2 shows the different configurations for the MCS-96 family of microcontrollers.

The following sections briefly describe some of the features of the MCS-96 microcontroller.

High Speed I/O Unit (HSIO)

The HSIO unit consists of the High-Speed Input unit (HSI), the High-Speed Output unit (HSO), one counter, and one timer. The "high-speed" means that the units can perform functions based on the timers without CPU intervention. The HSI unit records times when events occur and the HSO unit triggers events at preprogrammed times. All actions within the HSIO units are synchronized to the timer or counter.

The HSI unit can detect transitions on any of its four input lines. When one occurs, it records the time from Timer 1 and which input lines made the transition. The time is recorded with a 2 μs resolution and is stored in an eight-level first-in-first-out buffer (FIFO). The unit can activate an interrupt when the holding register is loaded or the 6th entry to the FIFO has been made.

Table 1. MCS®-96 Microcontroller Features and Benefits Summary

Features	Benefits
16-Bit CPU	Efficient machine with higher throughput.
8K Bytes ROM	Large program space for more complex, larger programs.
232 Bytes RAM	Large on-board register file.
Hardware MUL/DIV	Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.5 μs @ 12 MHz.
6 Addressing Modes	Provides greater flexibility of programming and data manipulation.
High Speed I/O Unit 4 dedicated I/O lines 4 programmable I/O lines	Can measure and generate pulses with high resolution (2 μs @ 12 MHz).
10-Bit A/D Converter	Reads the external analog inputs.
Full Duplex Serial Port	Provides asynchronous serial link to other processors or systems.
Up to 40 I/O Pins	Provides TTL compatible digital data I/O including system expansion with standard 8 or 16-bit peripherals.
Programmable 8 Source Priority Interrupt System	Respond to asynchronous events
Pulse Width Modulated Output	Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.
Watchdog Timer	Provides ability to recover from software malfunction or hardware upset.
48 Pin (DIP) & 68-Pin (Flatpack, Pin Grid Array) Versions	Offers a variety of package types to choose from to better fit a specific application need for number of I/Os and package size.

The HSO unit can be programmed to set or clear any of its six output lines, reset timer 2, trigger an A/D conversion, or set one of four software timer flags at a selected time. An interrupt can be enabled for any of these events and either Timer 1 or Timer 2 can be referenced for the programmed time value. Also, up to eight commands for preset actions can be stored in the Content Addressable Memory (CAM) file. After each action is carried out at the preset time, the command is removed from the CAM, making room for another command. The CPU is kept informed with a status bit that indicates if there is room for another command in the CAM.

A/D Converter

The analog-to-digital (A/D) converter is a 10-bit, successive approximation converter with an internal sample and hold circuit. It has a fixed conversion time of 88 CPU state times. A state time is one complete crystal frequency period. With a 12 MHz crystal, a state time is 250 nanoseconds (ns) so the conversion will take 22 μs.

The analog input needs to be in the range of 0 to VREF (nominally VREF = 5V) and can be selected from any of the eight analog input lines. The conversion is then initiated by either setting the control bit in the A/D command register or by programming the HSO unit to trigger the conversion at some specified time.

Serial Port

The on-chip serial port is compatible with the MCS-51 family (8051, 8031, etc.) serial port. It is a full duplex port and there is double-buffering on receive. Additionally, the serial port supports three asynchronous modes and one synchronous mode of operation. With the asynchronous modes eight or nine bits of data can be selected and even parity can optionally be inserted for one of the data bits. Selectable interrupts for transmit ready, receive ready, ninth data bit received, and parity error provide support for a variety of interprocessor communications protocols.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. The input to the baud rate generator can come from either the XTAL1 or the T2CLK pins. The maximum baud rate provided by the generator in asynchronous mode is 187.5K baud and in synchronous mode is 1.5M baud.

Watchdog Timer

The watchdog timer is a 16-bit counter which, once started, is incremented every state time. The watchdog timer is optionally started, and once started it cannot be stopped unless the system is reset. To start or clear the watchdog timer simply write a 1EH followed by a 0E1H to the WDT register (address 0AH). If not cleared before it overflows, the watchdog timer will pull the RESET pin low for two state times, causing the system to be reinitialized. With a 12 MHz crystal, the watchdog timer will overflow after 16 milliseconds (ms).

The watchdog timer is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows or the timer assumes that an upset has occurred and activates the RESET pin. Since the watchdog timer cannot be turned off by software, the system is protected against the upset inadvertently disabling the watchdog timer. The watchdog timer has also been designed to maintain its state through power glitches on VCC. The glitches can be as low as 0V or as high as 7V for as long as 1 μs to 1 ms.

Pulse Width Modulator (PWM)

The PWM output can produce a pulse train having a fixed period of 256 state times and a programmable width of zero to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM control register.

Table 2. Configurations of the MCS®-96 Family of Microcontrollers

Options		68 Pin	48 Pin
Digital I/O	ROMLESS	8096	8094
	EPROM	8796	8794
	ROM	8396	8394
Analog and Digital I/O	ROMLESS	8097	8095
	EPROM	8797	8795
	ROM	8397	8395

Memory Space

The addressable memory space of the MCS-96 microcontroller consists of 64K bytes. Although most of this space is available for general use, some locations have special purposes (0000H through 00FFH and 1FFEH through 207FH). All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in Figure 2.

The internal register locations (0000H through 00FFH) on the 8096 are divided into two groups, a register file and a set of Special Function Registers (SFRs).

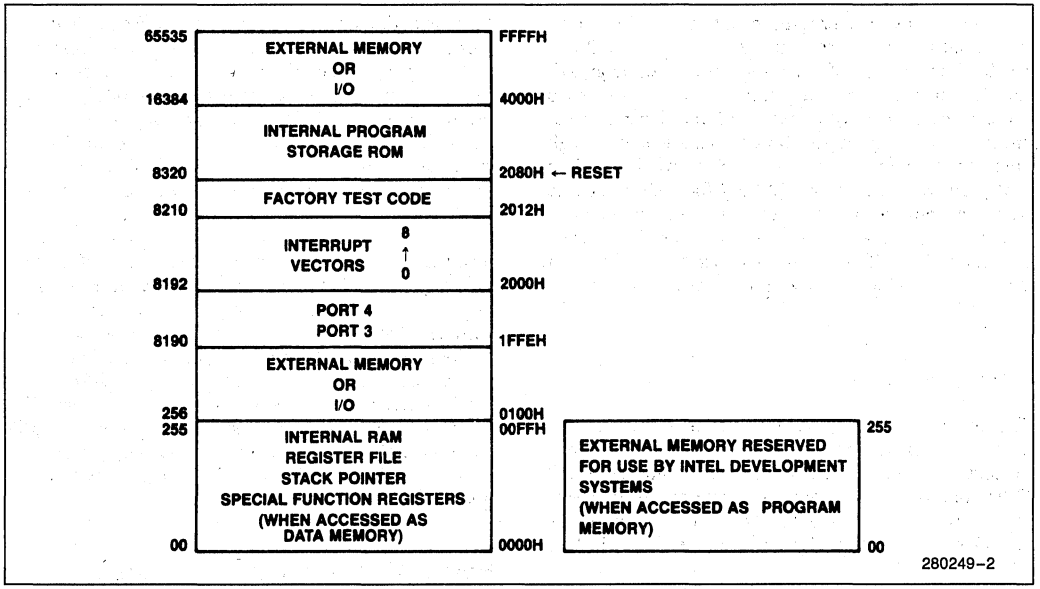

Figure 2. Memory Map

REGISTER FILE

Locations 1AH through 0FFH contain the register file. The register file memory map is shown in Figure 3. Additionally, locations 0F0H through 0FFH can be powered separately so that they will retain their contents when power is removed from the 8096 VCC pin. There are no restrictions on the use of the register file except that code cannot be executed from it. If an attempt to execute instructions from locations 00H through 0FFH is made, the instructions will be fetched from external memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI is also reserved for Intel development tools.

SPECIAL FUNCTION REGISTERS (SFRs)

Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. All of the I/O on the 8096 is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 3 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown in Figure 4. Note that these registers can be accessed only as bytes unless otherwise indicated. The stack pointer must be initialized by the user program and can point anywhere in the 64K memory space. The stack builds down, that is, it is a post-increment (POP), pre-decrement (PUSH) stack.

RESERVED MEMORY SPACES

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This enables easy reconstruction of these ports if external memory is used in the system. This also simplifies changing between the ROMless, EPROMed, and ROMed parts without changing the program addresses for ports 3 and 4. If ports 3 and 4 are not going to be reconstructed, these locations can be treated as any other external memory location.

The nine interrupt vectors are stored in locations 2000H through 2011H. The ninth vector (2010H–2011H) is reserved for Intel development systems. Figure 5 shows the interrupt vector locations and priority. When enabled, an interrupt occurring on any of these sources will force a call to the location stored in the vector location for that interrupt source. Internal locations 2012H through 207FH are reserved for Intel's factory test code and for use by future components. To ensure compatibility with future parts, external locations 2012H through 207FH (if present) should contain the hex value FFH.

SOFTWARE DEVELOPMENT OVERVIEW

MCS®-96 Microcontroller Software Development Packages

The MCS-96 Microcontroller Software Support Package provides 8096 development system support specifi-

cally designed for the MCS-96 family of single chip microcontrollers. The package consists of a symbolic macro assembler (ASM-96), Linker/Relocator (RL-96), Floating Point Arithmetic Library (FPAL96) and the librarian (LIB-96).

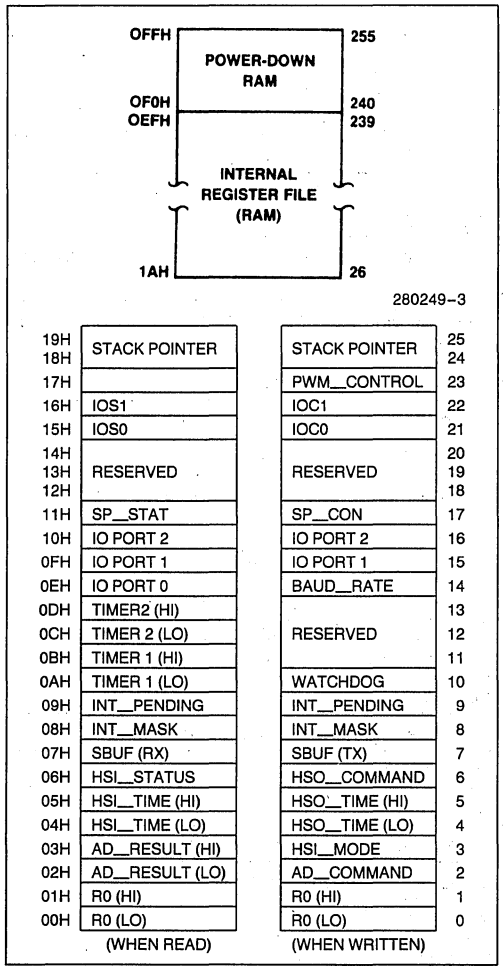

Figure 3. Register File Memory Map

The PL/M-96 Software Package provides 8096 high-level language development system suport. The package consists of a structured high-level language compiler (PL/M-96), Linker/Relocator (RL-96), Floating Point Arithmetic Library (FPAL96) and the librarian (LIB-96).

Both software packages run on the IBM PC XT and AT (with DOS 3.0 or greater) and on Series III/IV Intellec® development systems.

A detailed description of the tools contained in the packages is given in the following sections.

ASM-96 MACRO ASSEMBLER

The 8096 macro assembler translates the symbolic assembly language instructions into relocatable object code. Since the object modules are linkable and locatable, ASM-96 encourages modular programming practices. The macro facility in ASM-96 enables programmers to save development and maintenance time, since common code sequences only have to be done once. The assembler also provides conditional assembly capabilities. ASM-96 supports symbolic access to the many features of the 8096 architecture as described previously. A file is provided with all of the 8096 hardware registers defined. Alternatively, the user can define any subset of the 8096 hardware register set. Math routines are supported with instructions for 16 x 16-bit multiply or 32/16-bit divide.

Modular programs divide a rather complex program into smaller functional units that are easier to code, to debug, and to change. The separate modules can then be linked and located as desired into one program module of executable code. Standard modules can be developed and used in different applications thus saving software development time.

PL/M-96

PL/M-96 is a structured, high-level programming language used for developing software for the Intel MCS-96 family of microcontrollers. Symbolic access to the on-chip resources of the MCS-96 microcontroller is provided in PL/M-96. The PL/M-96 compiler translates the PL/M-96 language into 8096 relocatable object code, compatible with object code generated by other MCS-96 translators (such as ASM-96). This enables improved programmer productivity and application reliability. PL/M-96 has been efficiently designed to map into the machine architecture, so as not to trade off higher programmer productivity with inefficient code. PL/M-96 is also compatible with PL/M-86 thus assuring design portability and minimal learning effort for programmers already familiar with PL/M.

COMBINING PL/M-96 AND ASM-96

For each procedure activation (CALL statement or function reference) in the source, the object code uses a calling sequence. The calling sequence places the procedure's actual parameters (if any) on the stack, then activates the procedure with a CALL instruction. The parameters are placed on the stack in left to right order. Since the direction of stack growth is from higher locations to lower, the first parameter occupies the highest position on the stack and the last parameter occupies

Register	Description
R0	Zero Register—Always read as a zero, useful for a base when indexing and as a constant for calculations and compares.
AD_RESULT	A/D Result Hi/Low—Low and high order Results of the A/D converter (byte read only)
AD_COMMAND	A/D Command Register—Controls the A/D
HSI_MODE	HSI Mode Register—Sets the mode of the High Speed Input unit.
HSI_TIME	HSI Time Hi/Lo—Contains the time at which the High Speed Input unit was triggered. (word read only)
HSO_TIME	HSO Time Hi/Lo—Sets the time for the High Speed Output to execute the command in the Command Register. (word write only)
HSO_COMMAND	HSO Command Register—Determines what will happen at the time loaded into the HSO Time registers.
HSI_STATUS	HSI Status Registers—Indicates which HSI pins were detected at the time in the HSI Time registers.
SBUF (TX)	Transmit buffer for the serial port, holds contents to be output.
SBUF (RX)	Receive buffer for the serial port, holds the byte just received by the serial port.
INT_MASK	Interrupt Mask Register—Enables or disables the individual interrupts.
INT_PENDING	Interrupt Pending Register—Indicates when an interrupt signal has occurred on one of the sources.
WATCHDOG	Watchdog Timer Register—Written to periodically to hold off automatic reset every 64K state times.
TIMER1	Timer 1 Hi/Lo—Timer 1 high and low bytes. (word read only)
TIMER2	Timer 2 Hi/Lo—Timer 2 high and low bytes. (word read only)
IOPORT0	Port 0 Register—Levels on pins of port 0.
BAUD_RATE	Register which contains the baud rate, this register is loaded sequentially.
IOPORT1	Port 1 Register—Used to read or write to Port 1.
IOPORT2	Port 2 Register—Used to read or write to Port 2.
SP_STAT	Serial Port Status—Indicates the status of the serial port.
SP_CON	Serial port control—Used to set the mode of the serial port.
IOS0	I/O Status Register 0—Contains information on the HSO status.
IOS1	I/O Status Register 1—Contains information on the status of the timers and of the HSI.
IOC0	I/O Control Register 0—Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.
IOC1	I/O Control Register 1—Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.
PWM_CONTROL	Pulse Width Modulation Control Register—Sets the duration of the PWM pulse.

Figure 4. SFR Summary

the lowest position. Note that a BYTE or SHORTINT parameter value occupies two bytes on the stack, with the value in the lower (even address) byte. The contents of the higher byte are undefined. A parameter of type WORD or INTEGER (16 bits) is pushed as a word. A parameter of type DWORD, LONGINT or REAL (32 bits) is pushed as two words; the high-order word is pushed first.

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 5. Interrupt Vector Locations

After the parameters are passed, the CALL instruction places the return address on the stack. Function results are returned via a global PL/M-96 double-word register, PLM$REG located at 1CH. If a byte value is returned, the low-order byte is used; if a word value is returned, the low-order word is used; otherwise, the full register is used. PL/M-96 uses the eight byte registers at addresses ICH–23H for temporary computations. The library PLM-96LIB defines the public symbol PLM$REG.

Table 3 describes symbol type matching between a PL/M-96 global variable and an ASM-96 global variable. Note that except for NULL, no matches occur between any ASM-96 type stamp and the PL/M-96 type stamps ARRAY and STRUCTURE. A mismatch warning can be prevented by attaching the type stamp NULL to the variable in question in the ASM-96 module.

The easiest way to ensure compatibility between PL/M-96 programs or procedures and ASM-96 subroutines is simply to write a dummy procedure in PL/M-96 with the same argument list as the desired assembly language subroutine and with the same attributes. Then, compile the dummy procedure with the specified CODE control. This will produce a pseudo-assembly listing of the generated MCS-96 code, which can then be copied as the prologue and epilogue of the assembly language subroutine.

OTHER SOFTWARE DEVELOPMENT TOOLS

The RL96 linker and relocator program is a utility that performs two functions useful in MCS-96 software development. First, the link function combines a number of object modules generated by ASM-96, PL/M-96, and system libraries (such as PLM96.lib and FPAL96.lib) into a single program. Secondly, the locate function assigns an absolute address to all relocatable addresses in the linked MCS-96 object module. RL96 resolves all external symbol references between modules and will select object modules from library files if necessary. besides the absolute object module file, RL96 produces a listing file that shows the results of the link/locate, including a memory map symbol table and an optional cross reference listing. With the relocator the programmer can concentrate on software functionality and not worry about the absolute addresses of the object code. All program symbols are passed through into the object module as debug records. The FPAL96 floating point arithmetic library contains single precision 32-bit floating point arithmetic functions. All math complies with the IEEE floating point standard for accuracy and reliability. FPAL96 includes the basic arithmetic operations (i.e., add, subtract, multiply, divide, mod, square root) and other widely used operations (i.e., compare, negate, absolute, remainder). An error handler is included to handle exceptions commonly encountered during arithmetic operations such as divide by zero.

The LIB96 utility creates and maintains libraries of software object modules. The user can develop standard modules and place them in libraries. Application programs can then call these modules using predefined interfaces. LIB96 has a streamlined set of commands (create, add, delete, list, exit) to provide ease of use. When using object libraries, RL906 will only include those object modules that are required to satisfy external references, thus saving memory space.

Table 3. ASM96-PL/M-96 Symbol Type Matching

PL/M-96 / ASM96	Byte	Word	Dword	Short Int	Integer	Long Int	Real	Array	Structure	Label	Procedure
BYTE	M			M							
WORD		M			M						
LONG			M			M					
REAL							M				
ENTRY										M	M
NULL	M	M	M	M	M	M	M	M	M	M	M

iSBE-96 EMULATOR OVERVIEW

Introduction to the iSBE-96 Emulator

The iSBE-96 Single Board Emulator supports the execution and debugging of programs for the MCS-96 family of microcontrollers at speeds up to 12 MHz. Figure 6 shows a block diagram of the iSBE-96 emulator. The iSBE-96 emulator consists of an 8097 microcontroller, a 12 MHz execution clock, 16K of zero wait state RAM memory, and a user cable which connects the MCS-96 pin functions to the user's prototype system. The iSBE-96 emulator also supports an 8096 extended address/data bus for users with off chip memory and reconstructs port 3 and 4 for the users of the ROMed parts, 839x, and the EPROM parts, 879x. Additionally, the iSBE-96 emulator provides two RS-232 serial ports, serial communications cable, an EPROM based monitor for fundamental emulator control and functionality, and a software program for interfacing to a host computer. Intel currently supports an IBM PC XT and AT, and the Series III/IV Intellec development systems as hosts.

iSBE-96 Emulator I/O

The iSBE-96 emulator's on-board input and output (I/O) devices are used to manage the emulator's resources. These I/O devices are mapped into memory at locations 1F00H through 1FFFH. This memory block (1F00H through 1FFFH) is reserved for use by the iSBE-96 emulator. Table 4 shows the iSBE-96 memory mapped I/O address assignments. Since this memory block is in all possible memory configurations of the iSBE-96 emulator (see Figure 7 for the iSBE-96 memory map), it is possible for user programs to utilize any or all of the system I/O devices.

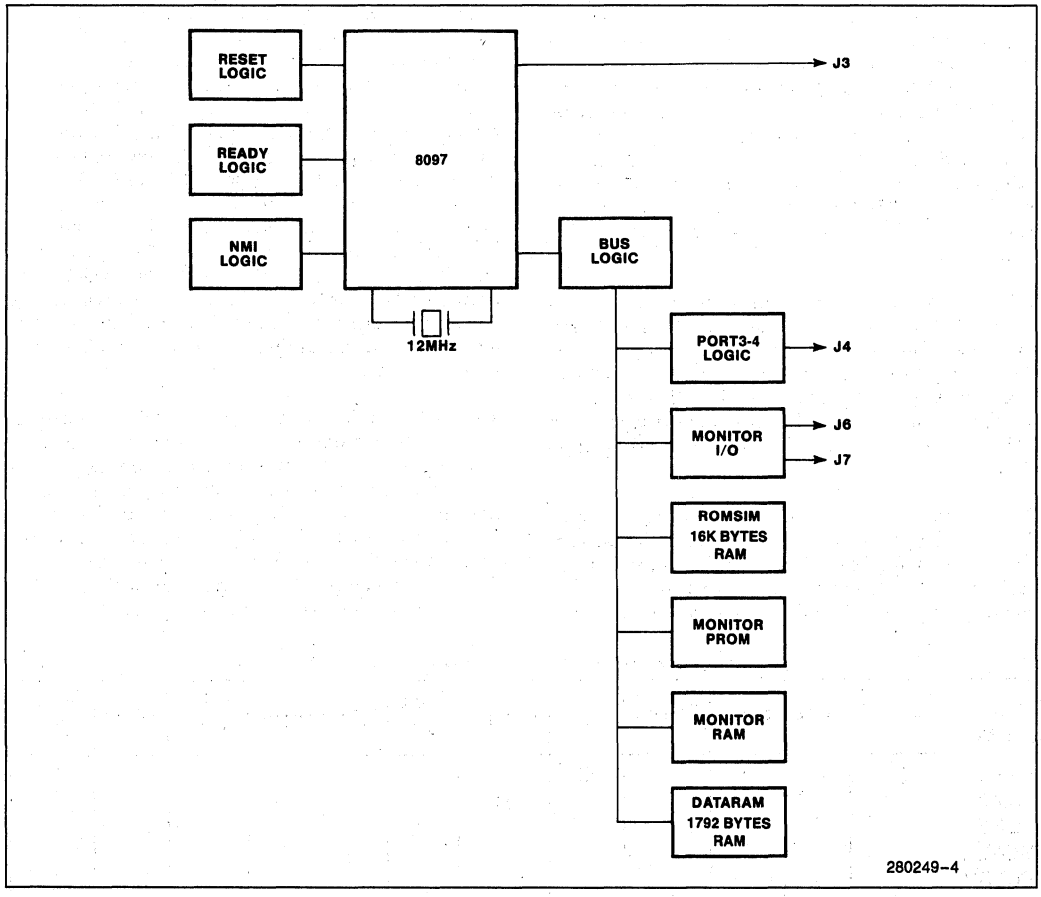

Figure 6. Block Diagram for the iSBE-96 Single Board Emulator

Table 4. iSBE-96 Memory Mapped I/O Address Assignments

Address	Function
01FEO	Data set USART data register
01FE2	Data set USART control/status register
01FE4	Data terminal USART data register
01FE6	Data terminal USART control/status register
01FE8	Timer counter 0
01FEA	Timer counter 1
01FEC	Timer counter 2
01FEE	Timer mode control register
01FF0	iSBE-96 mode register
01FF2	Port 3/4 control register
01FFE	Port 3 reconstruction
01FFF	Port 4 reconstruction

RS-232 SERIAL PORTS

Included as part of the on-board I/O are two RS-232 serial ports. One is configured as Data Communications Equipment (DCE) and the other as Data Terminal Equipment (DTE). When operating with the host software provided with the iSBE-96 emulator, the DCE port is used for the system console and the link for exchanging files. Table 5 shows the pin configuration of the two serial port connectors.

The serial ports are serviced under control of the on-board 8097 non-maskable interrupt (NMI). The NMI has the highest priority of all interrupts on the 8097 microcontroller. While in emulation (user program is executing) the user program will be interrupted if monitor commands are entered from the console. Valid commands input on the console will be executed by the monitor even during emulation. Therefore, the iSBE-96 emulator provides full-speed 12 MHz emulation, only if no commands are entered until emulation is halted.

MCS®-96 PORT 3/4 AND EXTENDED ADDRESS/DATA BUS

With the MCS-96 microcontroller, ports 3 and 4 pins can be used as actual port pins or as an extended address and data bus. For the convenience of the users of the ROMed parts and the EPROMed parts (839x and 879x respectively) the iSBE-96 emulator provides a reconstruction of ports 3 and 4. Additionally, for users of the ROMless parts or parts in external access mode, the iSBE-96 emulator provides an extended address and data bus. The selection of what the port pins are used for is left to the user via the MAP BUSPINS command. On power-up of the iSBE-96 emulator, the default mapping is for port 3/4.

iSBE-96 Emulator Memory Map

The iSBE-96 emulator has a number of memory map options. All of the memory maps are compatible with the MCS-96 microcontroller. Figure 7 shows the different memory map selections available. Each memory map is selected by the MAP MODE command, which changes the memory map currently recognized by the iSBE-96 emulator. Table 6 summarizes the physical memory configurations of the iSBE-96 emulator needed to implement the various memory maps. Note that modes (memory maps) 1 through 3 require that the eight 2K x 8 RAM chips (16K bytes of RAM) on the iSBE-96 emulator be replaced by 8K x 8 RAM or PROM chips.

The memory map is controlled by two bipolar PROMs and an eight bit register (the mode register at 01FF0H). The format of the mode register is shown in Figure 8. The mode register is a write only register and any writes to this register need to be done with caution. In addition to the memory map, the mode register is used to enable each of the five possible sources of interrupts connected to the NMI.

Monitor Command Summary

The iSBE-96 monitor is capable of executing a number of commands without being connected to a host development system. It is possible to connect only a video terminal to the iSBE-96 emulator and still have significant debug capability. Table 7 summarizes the monitor commands. The load and save command will not work with the iSBE-96 emulator connected to a terminal. Load and save requires the iSBE-96 emulator to be connected to a host development system. If a non-Intel suported host is used a software program will need to be written for that computer to provide the mass storage/retrieval access and the proper communications interface protocol to the iSBE-96 emulator.

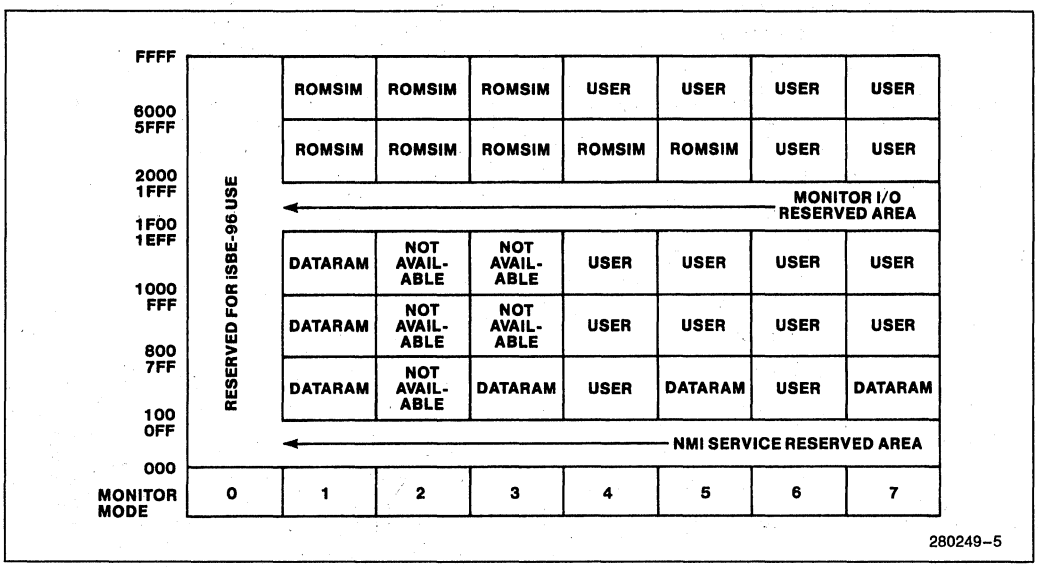

Figure 7. iSBE-96 Memory Map and Monitor Modes

Design Considerations

When debugging MCS-96 designs with the iSBE-96 emulator, there are some features of the emulator that should be considered or taken into account as early in the design process as possible.

MEMORY

The user's prototype memory should be mapped to be compatible with one of the iSBE-96 memory maps (illustrated in Figure 7) or else a new memory map for the iSBE-96 emulator must be generated. External address locations 0000H through 00FFH and locations 1F00H through 1FFFH are reserved for development system use and should not be used when using an Intel emulator.

Program code or memory mapped peripherals should be temporarily relocated before debugging with the iSBE-96 emulator.

Table 5. DS/DT RS-232 Pin-Out Configuration

Pin Number	Signal Name/Connector	
	DCE/J7	DTE/J6
1	GND	GND
2	TXD-I	TXD-O
3	RXD-O	RXD-I
4	RTS-I	RTS-O
5	CTS-O	CTS-I
6	DSR-O	DSR-I
7	GND	GND
20	DTR-I	DTR-O

Table 6. Memory Configurations for Each Mode

Mode	Allowable Memory Configurations
0	Monitor
1	8K x 8 Static RAMs or PROMs installed
2	8K x 8 Static RAMs or PROMs installed
3	8K x 8 Static RAMs or PROMs installed
4	User prototype may be RAM or PROM
5	User prototype may be RAM or PROM
6	All memory is on prototype, RAM or PROM
7	All memory above 7FFH is on prototype, RAM or PROM

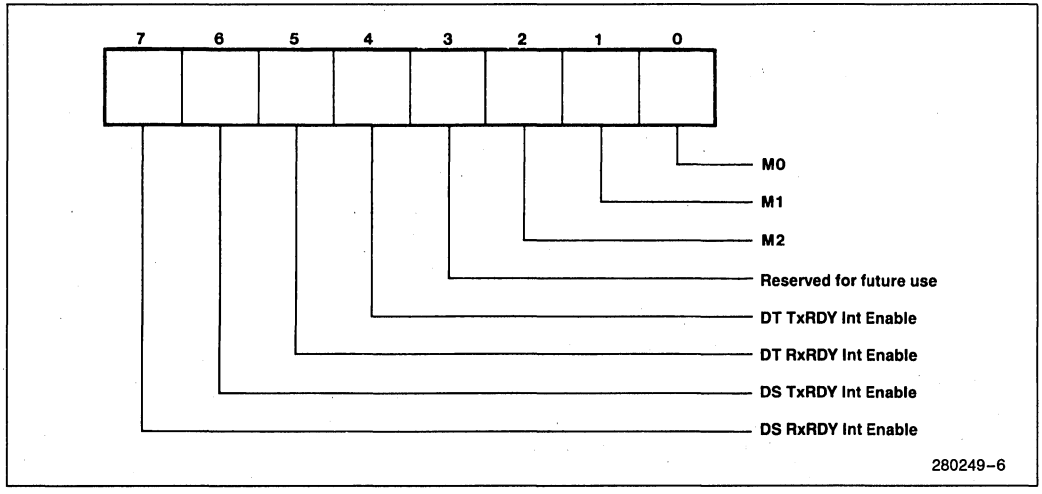

Figure 8. MODE Register Format

BREAKPOINTS

When emulation breakpoints or single-step emulation is used, the iSBE-96 monitor requires six bytes of the user's stack space. Since the ASM-96 assembler and the PL/M-96 compiler do not automatically take this into account, an extra six bytes of stack space needs to be allocated either explicitly in the code or implicitly with the STACKSIZE control of RL-96.

Since the trap vector (locations 2010H and 2011H) is utilized by the iSBE-96 emulator to provide breakpoints in emulation and single-step emulation, the trap vector locations must remain in RAM space or breakpoints and single stepping will not work. The iSBE-96 emulator could still go into emulation if these locations are in ROM or EPROM, but the ability to set breakpoints and single-step would be lost. In this case, emulation would be halted by sending an escape (<esc>) command to the iSBE-96 emulator.

When breakpoints are set, the instruction at the breakpoint is executed in single-step mode and not in real time. All other instructions up to the breakpoint are executed in real time. Here is one example of how the implementation of breakpoints affects debugging programs. Normally, a break on a PUSHF instruction at the start of a low priority interrupt service routine should enable the service routine to continue executing when emulation is resumed. Because the last instruction at the breakpoint is executed in non-real time, a higher priority interrupt could occur before the PUSHF instruction is actually executed. If this is the case, the higher priority interrupt would be serviced before the breakpoint at the PUSHF instruction. The breakpoint should be set on the instruction after the PUSHF if the higher priority interrupts need to be disabled.

MCS®-96 MICROCONTROLLER INTERRUPTS

All interrupt vector locations (2000H–200EH) should be initialized. This is a good practice even if the iSBE-96 emulator is not used for debug. This will prevent a system lock-up or crash in the event that the program unexpectedly enables interrupts. The vectors contain random addresses upon power up since the default memory map for the vector locations is in RAM. When a breakpoint is encountered during emulation, or while single-stepping, the monitor temporarily writes a trap instruction (0F7H) at all locations stored in the interrupt vectors. This could have adverse effects if the vector happened to contain the address of a register location, program data location or an instruction operand.

Any of the 8097 programmed events based on timer 1, timer 2 or external interrupts will continue to occur even while emulation of the iSBE-96 emulator has been stopped. When resuming emulation, these interrupts may be pending and would be serviced in order of priority. This could possibly cause an endless loop of service routines, overflow of the stack or differences between real-time emulation and emulation with breakpoints. Any code involving real-time events that has been debugged using breakpoints or single-step emulation should be verified in full speed, non-interrupted emulation.

Table 7. iSBE-96 Monitor Commands

Monitor Command	Function
BAUD	Sets up the baud rate.
BR	Enables display and setting of up to eight software breakpoints.
BYTE	Enables display and changing of a single byte or range of bytes of memory or a single byte of the 8097 internal registers.
CHANGE	Enables display and changing of a series of memory words or bytes.
<CONTROL>S	Stops scrolling of the screen display.
<CONTROL>Q	Resumes scrolling of the screen display.
<CONTROL>X	Deletes the line being entered.
<ESCAPE>	Aborts the command executing.
GO	Begins emulation and continues until an enabled breakpoint is reached or the escape key is pressed.
LOAD	Loads programs and data from disks.
MAP	Enables mapping of several preprogrammed memory maps; also enables configurable serial I/O and selective servicing of the watchdog timer.
PC	Displays and changes the program counter.
PSW	Displays and changes the program status word.
RESET CHIP	Resets the 8096 to power-up conditions.
SAVE	Saves programs and data to disks.
SP	Displays and changes the stack pointer.
STEP	Provides single-step emulation with selective display formats.
VERSION	Displays the monitor version number.
WORD	Enables display and changing of a single word or range of words of memory or a single word of the 8097 internal registers.

MCS®-96 Microcontroller Port 3/4

For anyone reconstructing port 3 and 4 (1FFEH and 1FFFH) on their target system, more care must be taken to debug the system. Since part of the port 3/4 reconstruction is an address decoder for 1FFEH and 1FFFH, the easiest thing to do is to temporarily change the mapped address for port 3/4 out of the reserved memory block. This means that both the hardware as well as the software has to be modified, but this enables debugging the integrated hardware and software. The software could automatically change the port addresses for debugging with the use of conditional assemble or compile statements.

The other method for debugging port 3/4 requires that the hardware and software be debugged separately or at least in stages. The user system, except for the port 3/4 reconstruction and any code utilizing port 3/4, would have to be debugged first. Then, with the iSBE-96 emulator in port 3/4 configuration (using MAP BUSPINS = PORT 34), the iSBE-96 emulator would be connected directly to the user's system port 3/4 pins. That is, the iSBE-96 port 3/4 pins on connector J4 would be connected on the port side of the user's port 3/4 reconstruction, bypassing it altogether.

CONNECTING THE iSBE-96 EMULATOR TO THE IBM PC XT AND AT

Introduction

A communications program (driver) is supplied with the iSBE-96 emulator so that it can be used with an

IBM PC XT and AT, as well as an Intel Series III or Series IV development system. This driver provides an enhanced command set (extensions shown in Table 8) for the iSBE-96 emulator and provides access to the host system's mass storage.

The following sections describe the additional features provided by the driver.

iSBE-96 Emulator Additional Commands Available

In addition to the command set provided by the iSBE-96 monitor, the driver provides a set of computer system interface commands. The additional commands provided by the driver are summarized in Table 8. The driver provides the proper communications protocol to complete the implementation of the iSBE-96 monitor LOAD and SAVE command. The LIST command will save a copy of everything displayed on the console to a system file, creating a complete log of the emulation session for future reference. Also, the INCLUDE command will redirect command input to come from a system file.

iSBE-96 Emulator Symbolic Support

The iSBE-96 monitor supports the use of symbolics for the program counter (PC), program status word (PSW), and stack pointer (SP). Additionally, the driver supports symbolics for the MCS-96 special function registers in the ASM and DASM commands. With this feature, the symbolic reference can be to a special function register when using the ASM and DASM commands rather than the register address, which can be cumbersome to remember or look up. Figure 9 contains a list of the symbolics supported by the ASM and DASM commands. These symbols are compatible with the MCS-96 symbols listed in Figure 4.

MODIFYING THE iSBE-96 EMULATOR CLOCK SPEED

Introduction

Although it comes standard with a 12 MHz crystal, the iSBE-96 emulator is designed to operate at crystal frequencies from 6 MHz to 12 MHz. The iSBE-96 monitor power-up diagnostics include board-level serial port tests that take advantage of the 12 MHz crystal frequency. Therefore, to operate the iSBE-96 emulator at other crystal frequencies, it is necessary to disable the power-up diagnostics. Only two simple modifications are needed: altering the monitor code and changing the crystal itself.

iSBE-96 Monitor Patch

The first modification disables the power-up diagnostics. This is completed by changing the monitor's 3-byte CALL instruction to the diagnostics to NOP (no-operation) instructions. The call to diagnostics is located at

Table 8. Driver Commands

Driver Command	Function
ASM	Loads memory with translated MCS-96 assembler mnemonics.
DASM	Displays memory as MCS-96 assembler mnemonics.
EXIT	Exits the driver and returns to the host operating system.
<CONTROL>C	Same as for the EXIT command, but will not properly close the system serial port.
HELP	Displays the syntax of all commands.
INCLUDE	Specifies a command file.
<CONTROL>I	Turns the command file on and off.
<TAB>	Same as <CONTROL>I (turns the command file on and off).
LIST	Specifies a list file.
<CONTROL>L	Turns list file on and off.
<CONTROL>S	Stops scrolling of the screen display.
<CONTROL>Q	Resumes scrolling of the screen display.
<CONTROL>X	Deletes the line being entered.
<ESCAPE>	Aborts the command executing.

EPROM address 1046H (monitor address 208CH). The following is a step-by-step explanation of what to do to the monitor, version 1.1, to make the patch.

1. Remove the low-byte monitor EPROM (U53) and, using a PROM programmer, copy its contents to the PROM programmer data buffer.
2. Change bytes 1046H and 1047H in the data buffer from 0EFH and 32H, respectively, to 0FDH.
3. Using another 27128 EPROM with 250 nanosecond access time, program a new monitor PROM and install it in the iSBE-96 emulator as U53.
4. Remove the high-byte monitor EPROM (U61) and, using a PROM programmer, copy its contents to the PROM programmer data buffer.
5. Change byte 1046H in the data buffer from 2CH to 0FDH.
6. Using another 27128 EPROM with 250 nanosecond access time, program a new monitor PROM and install it in the iSBE-96 emulator as U61.

With this change in place the DIAGS LED on the iSBE-96 emulator will not go off after power-up. If for any reason you suspect a problem with the iSBE-96 emulator, reinstall the original monitor PROMs and use the power-up diagnostics for system checkout or before servicing the iSBE-96 emulator.

iSBE-96 Crystal Modification

There are now two ways to modify the iSBE-96 emulator to operate at different clock speeds. The first is by far easier and the second involves more work.

The first method of modifying the iSBE-96 emulator is to simply replace the 12 MHz crystal, Y1, with the desired crystal. The only restriction is that the new crystal must be between 6 MHz and 12 MHz.

The second method is to modify the iSBE-96 emulator to use the target system crystal frequency. To do this, carefully remove crystal Y1 and capacitors C6 and C7 from the iSBE-96 emulator. The target system crystal oscillator should be buffered with the circuit shown in Figure 10. The buffer output connects to the empty Y1 board connection closest to the edge of the board, as shown in Figure 11. The target system clock is also limited to 6 MHz through 12 MHz.

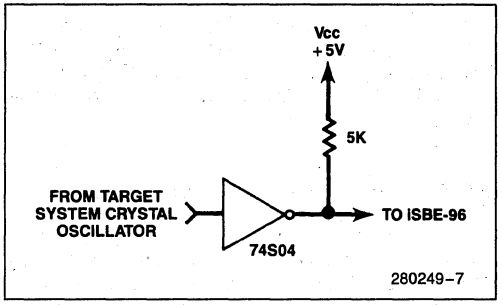

Figure 10. External Clock Drive

AD_COMMAND	IOPORT2
AD_RESULT	IOPORT3
AD_RESULT_HI	IOPORT4
AD_RESULT_LO	IOS0
BAUDRATE	IOS1
HSI_MODE	PWM_CONTROL
HSI_STATUS	SBUFRX
HSI_TIME	SBUFTX
HSI_TIME_HI	SP
HSI_TIME_LO	SP_CONN
HSO_COMMAND	SP_STAT
HSO_TIME	TIMER1
HSO_TIME_HI	TIMER1_HI
HSO_TIME_LO	TIMER1_LO
INT_MASK	TIMER2
INT_PENDING	TIMER2_HI
IOC0	TIMER2_LO
IOC1	WATCHDOG
IOPORT0	ZERO
IOPORT1	

Figure 9. ASM and DASM Command Symbol Support List

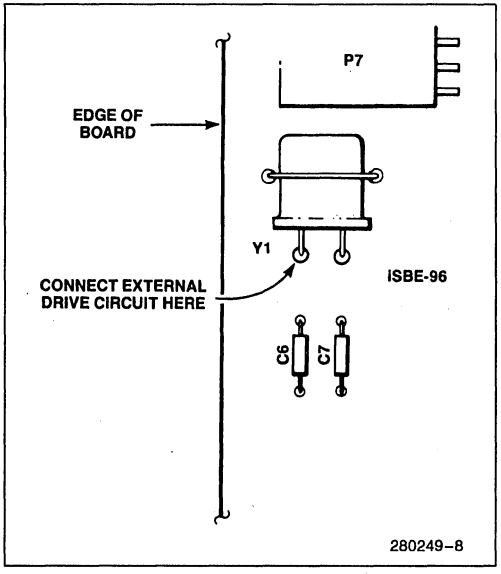

Figure 11. External Clock Connection

Care should be taken to ensure adequate digital ground connections between the target system and the iSBE-96 emulator. The user cable connected to J4 can be used for that purpose. All even numbered pins on J4 (except for pin 2) are connected to digital ground on the iSBE-96 emulator.

Before having your iSBE-96 emulator serviced by Intel, it should be restored to its original condition.

MODIFYING THE iSBE-96 MEMORY MAP

Introduction

The iSBE-96 emulator provides seven user memory map (mode) selections. There are eight total, but the monitor reserves the use of one map, mode zero. The iSBE-96 memory maps are illustrated in Figure 7. Even though these memory maps fulfill the majority of the user's needs, there will be times when a custom memory map is desired. This can be done easily if you follow the guidelines in this section.

The memory space for the MCS-96 microcontroller, as well as the 8097 used on the iSBE-96 emulator, has a range from 0 to 64K (0FFFFH) bytes. The 8097 has a linear memory space, but the data bus from the off-chip memory's even bytes are connected to the low eight data pins of the 8097 and the odd bytes are connected to the upper eight data pins. Therefore, if the memory map needs to be changed, it should be changed along even byte boundaries (2K, 4K, 16K, 32K) and should account for pairs of byte-wide memory chips (i.e., 2-2K x 8 and 2-8K x 8).

There are only two blocks of memory that have restrictions on them with the iSBE-96 emulator. These blocks are locations 0 through 0FFH and 1F00H through 1FFFH. These blocks are reserved for use by the iSBE-96 emulator and should always be mapped accordingly.

iSBE-96 Memory Map PROM

Before changing the iSBE-96 memory map PROM, it will help to know what it is and what it does.

The iSBE-96 memory map PROM (U39) is a 2K x 8 bipolar PROM. Since PROMs are one-time programmable, chances are that any changes will require replacement of the PROM. There is one key parameter when finding a replacement for the iSBE-96 memory map PROM, the time required from valid address on the input pins of the PROM to valid data on the output pins (t_{avdv}). The iSBE-96 memory map PROM requires a t_{avdv} time of 35 nanoseconds or better. An Intel 3636B-1 or any PROM satisfying the time requirements and having the standard JEDEC pin configuration can be used. Figure 12 shows the pin out and functional connections of the iSBE-96 memory map PROM.

Since the iSBE-96 memory map PROM is 2K bytes and there are eight memory maps, the memory map PROM is functionally segmented into eight blocks of 256 bytes each. Figure 13 illustrates the map PROM block assignments. Each block contains the map for one of the eight iSBE-96 monitor memory maps (modes) and each byte within a block contains the 'map' for 256 bytes of the total 64K byte address range. Figure 14 shows what the map byte contents should be to enable the different memory areas that are re-mappable.

The DATARAM (locations 100H through 7FFH) is not totally re-mappable. The DATARAM can be relocated to any 4K area in the 64K address range, but it always has to be at locations 100H through 7FFH in that 4K area.

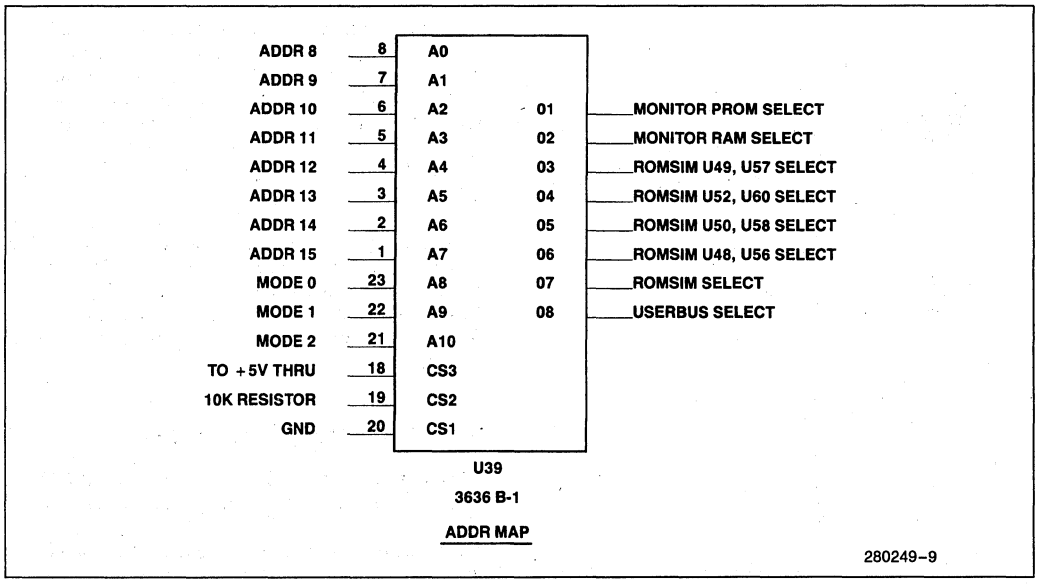

Figure 12. iSBE-96 Address Map PROM

MAP PROM Address	Monitor Memory Mode
0–0FFH	0
100–1FFH	1
200–2FFH	2
300–3FFH	3
400–4FFH	4
500–5FFH	5
600–6FFH	6
700–7FFH	7

Figure 13. MAP PROM Blocks

Chip Location	MAP Byte	Current MAPPED Address
U49–U57	0BBH	2000–2FFFH
U52–U60	0B7H	3000–3FFFH
U50–U58	0AFH	4000–4FFFH
U48–U56	9FH	5000–5FFFH
User	7FH	—
DATA RAM	0BDH	100–7FFH

Figure 14. iSBE-96 Map PROM Key

Sample iSBE-96 Memory Map Modification

As an example, let's say I have an iSBE-96 memory map that matches the map of the system I am developing. The map that I want needs to have locations 100H through 10FFH for mapped I/O devices, 1100H through 17FFH for scratch pad RAM, and 2000H through 0FFFFH for my EPROM application.

The I/O in my system is working, but I don't have the scratch pad RAM working yet and I don't want to program EPROMs until I have debugged my application program. So, what I really want is the scratch pad RAM mapped to iSBE-96 DATARAM and my EPROM memory area mapped to iSBE-96 RAM (ROMSIM). To accomplish the mapping for the EPROM, the iSBE-96 ROMSIM will have to be replaced by larger RAMs, as shown in Figure 15.

After looking at the different map modes (see Figure 7) I can see that mode 2 is close, but not quite it. So, mode 2 is the mode that I decide to change.

The following are the steps necessary to make the change.

1. Remove U48–U50, U52, U56–U58, and U60 on the iSBE-96 emulator.
2. Install 8K x 8, 150 nanosecond t_{avdy} static RAMs in their place and jumper the iSBE-96 emulator per Table B-2 in the iSBE-96 User's Guide, shown here as Table 9.
3. Remove the iSBE-96 map PROM (U39) and, using a PROM programmer, copy its contents to the PROM programmer data buffer.
4. Change bytes 100H through 10FFH to 7FH, and 1100H through 17FFH to 0BDH.
5. Program a new map PROM and install it as U39.

The new memory map could now be accessed by entering MAP MODE = 2 on the iSBE-96 console.

As you did with the monitor PROMs, the original address map PROM should be retained in case the iSBE-96 needs to be serviced by Intel.

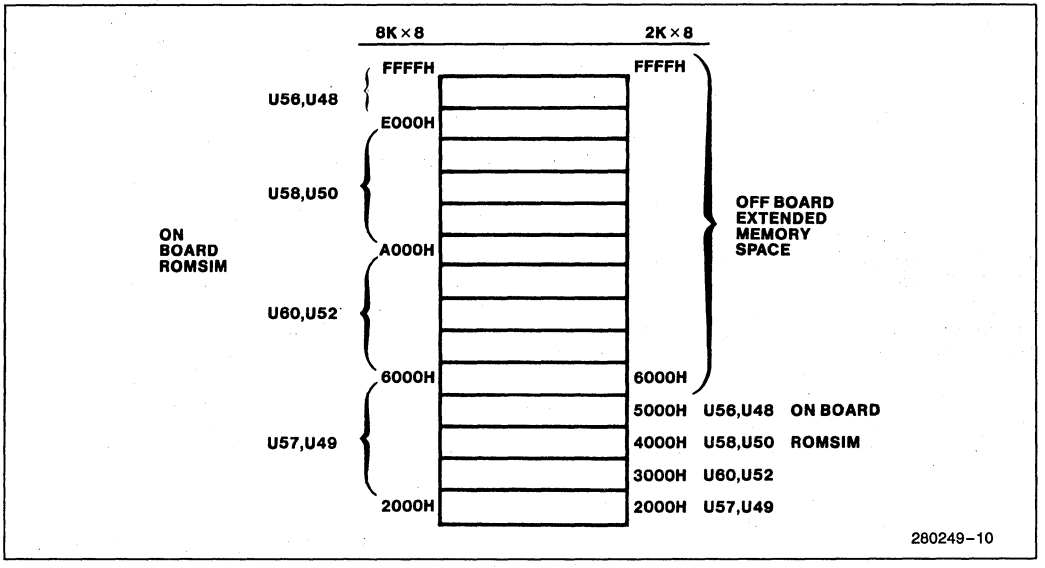

Figure 15. 8K x 8 Address Map

Table 9. 8K x 8 Replacement Jumper Configuration

Jumper Change		Function Incorporated by the Change
Default	Replacement	
E13–E14	E14–E15	Connects MA12 to U48
E16–E17	E17–E18	Connects MA12 to U49
E22–E23	E23–E24	Connects MA12 to U50
E31–E32	E32–E33	Connects MA12 to U52
E39–E40	E40–E41	Connects MA12 to U56
E47–E48	E48–E49	Connects MA12 to U57
E58–E59	E59–E60	Connects MA12 to U58
E77–E78	E78–E79	Connects MA12 to U60
E19–E20	OPEN	Disconnects U49, U57 pin 26 from VCC(1)
E36–E37	OPEN	Disconnects U48, U56 pin 26 from VCC(1)
E55–E56	OPEN	Disconnects U50, U58 pin 26 from VCC(1)
E74–E75	OPEN	Disconnects U52, U60 pin 26 from VCC(1)

NOTE:
1. It may be desirable to leave pin 26 connected to V_{CC}. Check pin out for 8K x 8 device used.

HELPFUL MCS®-96 PROGRAMS FOR THE iSBE-96

Introduction

During operation we discovered that the iSBE-96 emulator would be even more useful if it had a few more, or slightly different, commands. The following sections describe some helpful MCS-96 programs that can be used on the iSBE-96 emulator to make debugging your programs a little easier.

Memory Write Without Read Verify

As you may have already discovered, the iSBE-96 BYTE, WORD, and CHANGE commands do a read verify after writing the specified memory locations. This is very useful for determining if the memory is functioning, but requires that the memory be RAM. What then do you do if your system has memory mapped peripheral devices that access different registers for a read and write operation? The BYTE and WORD commands will write the location(s) correctly, but they will display a read verify error message.

Figure 16 illustrates an ASM-96 program that will perform the write to memory without a read verify. The program is located at 100H to correspond to the iSBE-96 DataRAM and thereby not intrude into user memory space. The program also uses eight bytes of internal 8097 register space. Again, so that the program does not intrude, the eight register bytes are pushed onto the stack and restored upon exit. You will have to ensure that there is sufficient stack available. The data structure containing the bytes and their respective addresses is assumed to be structured as follows: 150H byte containing the count of data bytes, 152H first data byte, 152H + byte count (+1 if byte count is odd) address for first data byte.

To use the program, first make sure you are in an iSBE-96 memory mode that provides DataRAM, then load the program object code. Once the program is loaded, put the data into the data structure at 150H: byte count, data bytes followed by data addresses. To execute the program simply type "GO FROM 100 TO 140". When the program stops at the breakpoint, the data bytes will have been written to the specified addresses.

Block Memory Move

If you have ever put something into memory and then decided that it should be located at another address, then you've probably wanted a block move program. It becomes tedious to move data structures or code a byte or word at a time. Sometimes it is inconvenient to relocate or link the original object code so that it can be loaded at the new location.

Since the MCS-96 instruction set utilizes relative offsets for the majority of the jump and branch instructions, it is feasible to move code blocks around. Of course, the block of code that you intend to move has to be either self-contained or small enough to fit within that mode of addressing. That is, the block of code moved should not contain a relative jump or branch to anywhere outside the block.

Figure 17 illustrates an ASM-96 program that will perform a block memory move. The program is located at 200H to correspond with the iSBE-96 DataRAM and so that it will not interfere with the write program described previously in "Memory Write Without Read Verify" section which is located at 100H. The program uses eight bytes of internal 8097 register space. So that the program is nonintrusive, the eight register bytes are pushed onto the stack and restored upon exit. You will have to ensure that there is sufficient stack available. The data structure containing the start, stop and destination addresses is assumed to be structured as follows: 230H start address, 232H stop address, and 234H destination address.

To use the program, first make sure you are in an iSBE-96 memory mode that provides DataRAM, then load the program object code. Once the program is loaded, put the data into the data structure at 230H: start address, stop address and finally destination address. To execute the program simply type "GO FROM 200 TO 22C". When the program stops at the breakpoint, the block of memory will have been moved to the specified location.

Writing/Reading an iSBE-96 Terminal in Emulation

There may be times while a program is executing that you would like to know how far it has progressed. But, you may not wish to use breakpoints to check the progress because they change the overall execution speed. This is particularly true for programs using real-time interrupts, since it may not be possible to use breakpoints. Since the iSBE-96 serial ports (DCE and DTE) are accessible during emulation, you can include program routines that write to a terminal or from the terminal to relay program status or dynamically change the program flow, provided you do it with care.

The iSBE-96 emulator uses the on-board 8097 NMI interrupt to service the DCE and DTE serial ports. This occurs even in emulation since there are some commands that are valid during emulation. Therefore, care should be taken when utilizing the unused serial port for dynamic program status. Since the iSBE-96 emulator is always connected to the host development system via the DCE serial port, a terminal can be connected to the unused DTE serial port. Incidentally, if you want to see what you're typing your program will need to echo it to the terminal.

```
MCS-96 MACRO ASSEMBLER        8096 Write with no Read Verify Routine        01/14/86

DOS MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: WRITE.A96
OBJECT FILE: WRITE.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: <none>

ERR  LOC   OBJECT        LINE        SOURCE STATEMENT
                            1  $TITLE ('8096 Write with no Read Verify Routine')
                            2
                            3
                            4  Write   MODULE  MAIN
                            5
     0100                   6          CSEG at 100h
                            7
     0100  C820             8  start:  push    20h             ;save working registers
     0102  C822             9          push    22h
     0104  C824            10          push    24h
     0106  C826            11          push    26h
     0108  B301500120      12          ldb     20h,150h        ;load byte count
     010D  990020          13          cmpb    20h,#0          ;make sure there are
                                                               ;bytes to write
     0110  DF26            14          je      J3
     0112  B10021          15          ldb     21h,#0          ;initialize registers
     0115  A1520122        16          ld      22h,#152h
     0119  C02420          17          st      20h,24h
     011C  302004          18          jbc     20h,0,J1        ;see if byte count is odd
     011F  65010024        19          add     24h,#1          ; if odd, add 1 for even
                                                               ;boundary
     0123  65520124        20  J1:     add     24h,#152h       ;load location of first byte
                                                               ;address
     0127  A22426          21  J2:     ld      26h,[24h]       ;load data byte address
     012A  B22321          22          ldb     21h,[22h]+      ;load data byte and
                                                               ;increment pointer
     012D  C62621          23          stb     21h,[26h]       ;write the byte
     0130  65020024        24          add     24h,#2          ;increment pointer to next
                                                               ;address
     0134  1520            25          decb    20h             ;done yet?
     0136  D2EF            26          jgt     J2
     0138  CC26            27  J3:     pop     26h             ;restore working registers
     013A  CC24            28          pop     24h
     013C  CC22            29          pop     22h
     013E  CC20            30          pop     20h
     0140  27FE            31  J4:     br      J4              ;wait here
                           32
     0142                  33  END
```

280249-11

Figure 16

```
MCS-96 MACRO ASSEMBLER    8096 Block Memory MOVE Routine                01/14/86

DOS MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: MOVE.A96
OBJECT FILE: MOVE.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: <none>

ERR  LOC   OBJECT       LINE   SOURCE STATEMENT
                          1   $TITLE ('8096 Block Memory MOVE Routine')
                          2
                          3
                          4   Move   MODULE  MAIN
                          5
     0200                 6          CSEG at 200h
                          7
     0200  C820           8   start: push    20h            ;save working registers
     0202  C822           9          push    22h
     0204  C824          10          push    24h
     0206  C826          11          push    26h
     0208  A301300220    12          ld      20h,230h       ;load start address
     020D  A301320222    13          ld      22h,232h       ;load end address
     0212  A301340224    14          ld      24h,234h       ;load destination address
     0217  882022        15   J1:    cmp     22h,20h        ;make sure there is
                                                             ;something to move
     021A  DE08          16          jlt     J2             ;if equal then only one
                                                             ;byte to move
     021C  B22126        17          ldb     26h,[20h]+     ;load byte and increment
                                                             ;source pointer
     021F  C62526        18          stb     26h,[24h]+     ;store byte and increment
                                                             ;destination pointer
     0222  27F3          19          br      J1             ;go see if done
     0224  CC26          20   J2:    pop     26h            ;restore working registers
     0226  CC24          21          pop     24h
     0228  CC22          22          pop     22h
     022A  CC20          23          pop     20h
     022C  27FE          24   J3:    br      J3             ;wait here
                         25
     022E                26   END

SYMBOL TABLE LISTING
--------------------

N A M E                                VALUE   ATTRIBUTES
J1. . . . . .                          0217H   CODE ABS ENTRY
J2. . . . . .                          0224H   CODE ABS ENTRY
J3. . . . . .                          022CH   CODE ABS ENTRY
MOVE. . . . . .                        -----   MODULE MAIN STACKSIZE(0)
START. . . . . .                       0200H   CODE ABS ENTRY

ASSEMBLY COMPLETED,      NO ERROR(S) FOUND.
```

280249-12

Figure 17

Figure 18 illustrates the PL/M-96 procedures to read and write a terminal connected to the DTE serial port on the iSBE-96 emulator and a sample calling program. The sample program uses an initial delay to ensure that the iSBE-96 NMI line has stabilized so that spurious NMI interrupts are not caused by accessing the DTE serial port. Figure 19 illustrates steps to compile and link the sample program.

To run the program, first load the sample program object code into the iSBE-96 emulator using the LOAD command. Then, type "GO FROM 2080 FOREVER". When you are ready to stop, press the escape key and emulation will halt.

iSBE-96 SERIAL PROTOCOL FOR LOAD AND SAVE

Introduction

The iSBE-96 emulator has a number of resident monitor commands, as described in Table 7. Normally, the iSBE-96 emulator is hosted by an IBM PC XT or AT, or an Intel Series III or Series IV development system. If you have a different host, you must write your own software program (driver) that meets the software handshaking protocol required by the iSBE-96 emulator. In that way, the resident monitor commands can be executed with any computer or terminal connected to the iSBE-96 DCE or the DTE serial ports.

The normal configuration is for the iSBE-96 emulator to be attached to a host computer system on the DCE port. Alternately, the iSBE-96 emulator can be attached to a terminal on the DTE port, which leaves the DCE port free to be connected to a computer. The terminal would be used to enter iSBE-96 debug commands (including LOAD and SAVE) and the computer used solely for loading and saving MCS-96 program files.

Whichever way you do it, the proper iSBE-96 serial port (DCE,DTE) needs to be mapped appropriately for loading, saving, and console connections. The MAP CONSOLE command is used to change the serial port connection for the console device. The iSBE-96 emulator will default to the port that the console is connected to at power up. The MAP SEND command is used to designate which serial port the iSBE-96 monitor uses for data transfer (sending) for the SAVE command. The MAP RECEIVE command is used to designate which serial port the iSBE-96 monitor uses for data transfer (receiving) for the LOAD command.

The next three sections describe the handshaking protocol used by the iSBE-96 emulator for loading and saving files. They provide sufficient information to write your own program to load and save programs with the iSBE-96 emulator.

Handshaking Characters

There are two characters that are used for control during the actual file transfer, EOF (1AH) and ESC (1BH). Determination that one of the two control characters has been encountered requires the use of a third character, DLE (10H). When transferred as data, the DLE, EOF and ESC characters must be prefixed by a DLE character. Additionally any data byte when ANDed with 7FH that yields one of the control characters (90H,9AH, and 9BH) also needs to be prefixed by a DLE. DLEs sent as prefixes should not be included in the byte count and should not be stored as data.

Loading Files

The following describes the protocol required by the iSBE-96 emulator for loading files. The following terminology is used: <cr> denotes a carriage return; Console is the terminal or computer mapped to the iSBE-96 CONSOLE device; Sender is the computer mapped to the iSBE-96 SEND device; Receiver is the computer mapped to the iSBE-96 RECEIVE device.

1. Console sends 'LOAD <cr>' to the iSBE-96.
2. iSBE-96 sends an XON (11H) to Console.
3. Sender sends up to 16,384 bytes and waits for iSBE-96 to send an XON (11H).
4. iSBE-96 processes the transferred bytes and sends an XON (11H) to Sender.
5. Steps 3 and 4 are repeated until the transfer is complete.
6. Sender sends an EOF (1AH) to iSBE-96.
7. iSBE-96 sends a prompt ('*') to Console.

If, during the transfer, the iSBE-96 emulator receives an unprefixed ESC (1BH) from the Sender or from the Console, the load is aborted and an ESC is sent to the Sender. The Sender should then respond with an XON (11H) to acknowledge the ESC.

If the end of file is reached at any time during the load, the transfer is terminated. The full 16,384 (16KH) bytes do not necessarily have to be transferred.

```
DOS PL/M-96 V1.0 COMPILATION OF MODULE SAMPLE
OBJECT MODULE PLACED IN TERMRW.OBJ
COMPILER INVOKED BY: C:\UDI\PLM96.EXE TERMRW.P96

                $title (' iSBE-96 Terminal Read/Write Sample Program')
                $optimize (3)
  1             sample: DO;
                /* local declarations */
  2     1       DECLARE   msg1(*)     BYTE    DATA(44H,61H,76H,65H,20H,53H,63H,68H,
                                                  6FH,65H,62H,65H,6CH,20H,69H,73H,
                                                  20H,47H,52H,45H,41H,54H),
                          msg2(*)     BYTE    DATA(0dH,0aH),
                          msg3(*)     BYTE    DATA(72H,69H,67H,68H,74H,3FH),
                          (I,char)    BYTE,
                          dt_data     ADDRESS AT (1FE4H),
                          dt_status   ADDRESS AT (1FE6H),
                          bell        LITERALLY     '07H';
                /* Procedure declarations */
  3     1       ci: PROCEDURE  BYTE  PUBLIC;
  4     2       DO WHILE ((dt_status AND 02H) = 0H);      /* wait till RxRDY */
  5     3           END;
  6     2       char = dt_data AND 7FH;
  7     2       RETURN char;
  8     2       END ci;
  9     1       co: PROCEDURE (char)     PUBLIC;
 10     2       DECLARE char     BYTE;
 11     2       DO WHILE ((dt_status AND 1) = 0 );         /* wait till TxRDY */
 12     3           END;
 13     2       dt_data = char;
 14     2       END co;
                /* Program starts here */
 15     1       CALL TIME(50);
 16     1       dt_status = 37H;                          /* clear any errors on the DTs 8251A
                                                           USART */
 17     1       CALL TIME(1);
 18     1       char = dt_data;                           /* clear the DT receive buffer */
 19     1       DO I = 1 TO LENGTH(msg1);
 20     2           CALL co(msg1(I-1));
 21     2           END;
 22     1       char = 'n';
 23     1       DO WHILE (char = 'n');
 24     2           DO I = 1 TO LENGTH(msg2);
 25     3             CALL co(msg2(I-1));
 26     3             END;
```

280249-13

Figure 18

```
 27      2          DO I = 1 TO LENGTH(msg3);
 28      3            CALL co(msg3(I-1));
 29      3            END;
 30      2          char = ci;
 31      2          CALL co(char);
 32      2          IF ((char< >'Y') AND (char< >'y')) THEN DO;
 34      3            char = 'n';
 35      3            CALL co(bell);
 36      3            END;
 37      2          END;

 38      1      DO WHILE 1;                          /* wait here when done */
 39      2          END;

 40      1      END sample;

MODULE INFORMATION:

  CODE AREA SIZE                    = 00DBH      219D
  CONSTANT AREA SIZE                = 001EH       30D
  DATA AREA SIZE                    = 0000H        0D
  STATIC REGS AREA SIZE             = 0003H        3D
  OVERLAYABLE REGS AREA SIZE        = 0000H        0D
  MAXIMUM STACK SIZE                = 0004H        4D
  60 LINES READ

PL/M-96 COMPILATION COMPLETE.     0 WARNINGS,     0 ERRORS
```

280249-14

Figure 18 (Continued)

```
C:\SBE96 >plm96 termrw.p96

DOS PL/M-96 COMPILER V1.0
Copyright Intel Corporation 1983
PL/M-96 COMPILATION COMPLETE.     0 WARNINGS,     0 ERRORS

C:\SBE96 >rl96 termrw.obj,plm96.lib to termrw.abs stacksize(16)

DOS MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation
RL96 COMPLETED,          0 WARNING(S),          0 ERROR(S)

C:\SBE96 >
```

280249-15

Figure 19

Saving Files

The following describes the protocol required by the iSBE-96 emulator for saving files. The following terminology is used: <cr> denotes a carriage return; partition denotes an address range, specified as 'address TO address'; Console is the terminal or computer mapped to the iSBE-96 CONSOLE device; Sender is the computer mapped to the iSBE-96 SEND device; Receiver is the computer mapped to the iSBE-96 RECEIVE device.

1. Console sends 'SAVE partition <cr>' to iSBE-96.
2. iSBE-96 sends an STX (02H) to Receiver.
3. Receiver acknowledges with an XON (11H) to iSBE-96.
4. iSBE-96 sends up to 16,384 bytes and waits for Receiver to send an XON (11H).
5. Receiver processes the transferred bytes and sends an XON (11H) to the iSBE-96.
6. Steps 4 and 5 are repeated until the transfer is complete.
7. iSBE-96 sends an EOF (1AH) to Receiver.
8. iSBE-96 sends a prompt ('*') to Console.

If, during the transfer, the iSBE-96 emulator receives an ESC (1BH) from the Receiver or from the Console, the load is aborted and an ESC is sent to the Receiver. The Receiver should then respond with an XON (11H) to acknowledge the ESC.

If the end of file is reached at any time during the load, the transfer is terminated. The full 16,384 (16KH) bytes do not necessarily have to be transferred.

SAMPLE DEBUG SESSION WITH THE iSBE-96 EMULATOR

The following sample program requires the use of PL/M-96, ASM-96, and an iSBE-96 emulator. It assumes the iSBE-96 DCE serial port is connected to an IBM PC XT or AT and a terminal is connected to the iSBE-96 DTE serial port. The terminal should be set for full-duplex and 9600 baud operation.

Sample Program Description

The MCS-96 program chosen for the sample debug session combines and utilizes many of the features described throughout this applications note and was designed to show as many of the iSBE-96 emulator's features as possible. The sample program uses both a PL/M-96 main module and an ASM-96 module and demonstrates how to link them together. The sample program also uses the terminal input/output procedures discussed in the Block Memory Move Section for input to the program and to display status in real-time. Finally, the program makes use of one of the MCS-96 software timers for basic program timing.

The PL/M-96 main module is illustrated in Figure 20. As shown, the main module contains local declarations, procedure declarations, and the mainline PL/M-96 program. Functionally, the program uses software timer 1 to keep a real time clock which is then displayed to the terminal connected to the iSBE-96 DT serial port. Initially the 'clock' is set by entering the current time through the terminal connected to the iSBE-96 DT port.

The ASM-96 module is shown in Figure 21. It contains the interrupt service routine for the software timer interrupt which actually does the timing for the 'clock'. It also defines all of the other MCS-96 interrupt vectors (2000H to 200FH) to help guard against program runaway and to avoid program anomolies when debugging with the iSBE-96 emulator.

Figure 22 illustrates the DOS batch file (CLOCK.BAT) used to compile, assemble, and link the sample program. The STACKSIZE(20H) control is added to the RL96 invocation to allow sufficient stack space for the sample program and the six bytes required by the iSBE-96 emulator. This batch file assumes that PL/M-96, ASM-96 and the utilities and libraries are located in a directory called 8096.DIR while the sample program modules and batch file are in the home directory. After entering the sample program modules and batch file using a word processor such as AEDIT, the sample program can then be assembled, compiled, and linked by typing CLOCK followed by an enter.

If a word processor other than AEDIT is used, you should insure that the word processor did not put an end of file character (IAH) at the end of the source code files since the Intel assemblers and compilers cannot handle it. It can be removed using the DOS copy/b command.

Sample Program Discussion

Before beginning the sample debug session it may be helpful to have a brief synopsis of what the sample program does and why. The MCS-96 software timers are incremented once every eight state times and the maximum count possible for the software timers is 65,535 (64KH). For a 12 MHz input crystal frequency, a state time is 250 ns. Therefore, one second can be expressed as: $1 = 1/(250E-9 * 8 * 65{,}535 * X)$ where X is the number of times the software timer completes the specified number of counts (time-outs). If you solve for X you will find that $X = 7.6295$. This tells us that we need seven time-outs at the maximum count and one time-out at a count of 41,254 (65,535 * 0.6295).

```
PL/M-96 COMPILER          iSBE-96 Sample Debug Program
DOS PL/M-96 V1.0 COMPILATION OF MODULE CLOCK
OBJECT MODULE PLACED IN CLOCK.OBJ
COMPILER INVOKED BY:    C:\UDI\PLM96.EXE CLOCK.P96

                $title (' iSBE-96 Sample Debug Program')
                $optimize (3)
  1             clock: DO;
                /* local declarations */
  2     1       DECLARE   bell              LITERALLY   '07H',
                          BS                LITERALLY   '08H',
                          FOREVER           LITERALLY   'WHILE 1',
                          FALSE             LITERALLY   '0',
                          TRUE              LITERALLY   'NOT FALSE',
                          BOOLEAN           LITERALLY   'BYTE',
                          msg1(*)           BYTE        DATA(0dH,0aH),
                          msg2a(*)          BYTE        DATA(0,0,':',0,0,':',0,0),
                          msg2(8)           BYTE        FAST,
                          msg3(*)           BYTE        DATA('set time - hh:mm:ss <cr>'),
                          (I,char)          BYTE,
                          seconds           BYTE,
                          minutes           BYTE,
                          hours             BYTE,
                          tick              BYTE        FAST    PUBLIC,
                          tock              WORD        PUBLIC,
                          count             BYTE        EXTERNAL,
                          count1            BYTE,
                          not$done          BOOLEAN,
                          not$first         BOOLEAN,
                          HSO_TIME          WORD        AT (04H),
                          HSO_CMD           BYTE        AT (06H),
                          INT_MASK          BYTE        AT (08H),
                          INT_PENDING       BYTE        AT (09H),
                          TIMER1            WORD        AT (0AH),
                          dt_data           ADDRESS     AT (1FE4H),
                          dt_status         ADDRESS     AT (1FE6H);
                /* Procedure declarations */
  3     1       ci: PROCEDURE     BYTE     PUBLIC;
  4     2       DO WHILE ((dt_status AND 02H) = 0H);          /* wait till RxRDY */
  5     3           END;
  6     2       char = dt_data AND 7FH;
  7     2       RETURN char;
  8     2       END     ci;
  9     1       co:     PROCEDURE (char)     PUBLIC;
 10     2       DECLARE     char     BYTE;
 11     2       DO WHILE ((dt_status AND 1) = 0 );            /* wait till TxRDY */
 12     3           END;
```

280249-16

Figure 20

```
13    2     dt_data = char;
14    2     END     co;
15    1     init$DT:    PROCEDURE     PUBLIC;
16    2     dt_status = 37H;                          /* clear any errors on the DTs
                                                        8251A USART */
17    2     CALL TIME(1);
18    2     char = dt_data;                           /* clear the DT receive buffer */
19    2     END     init$DT;
20    1     ascii:    PROCEDURE (value,dest$ptr)     PUBLIC;
21    2     DECLARE (value,temp)     BYTE,
                    dest$ptr     ADDRESS,
                    (dest     BASED     dest$ptr) (2)     BYTE;
22    2     value = SHL((value/10),4) + (value MOD 10);   /* convert to BCD */
23    2     temp = value;
24    2     dest(0) = SHR(temp,4) + 30H;              /* convert to ASCII decimal value */
25    2     dest(1) = (value AND 0FH) + 30H;
26    2     END     ascii;
27    1     print$msg1:     PROCEDURE;
28    2     DECLARE     I     BYTE;
29    2     DO I = 1 TO LENGTH(msg1);
30    3          CALL co(msg1(I-1));
31    3          END;
32    2     END     print$msg1;
            /* Program starts here */
33    1     CALL TIME(50);                            /* delay to insure iSBE-96 NMI line
                                                        is stable */
34    1     CALL init$DT;                             /* initialize DT serial port */
35    1     count,count1 = 0;                         /* initialize variables */
36    1     not$done = TRUE;
37    1     not$first,tick = FALSE;
38    1     seconds,minutes,hours = 0;
39    1     CALL movb(.msg2a,.msg2,LENGTH(msg2a));
40    1     CALL print$msg1;
41    1     DO I = 1 TO LENGTH(msg3);                 /* query for initial time */
42    2          CALL co(msg3(I-1));
43    2          END;
44    1     CALL print$msg1;
45    1     DO WHILE not$done;                        /* input initial time values */
46    2          char = ci;
47    2          IF ((char > = 30H) AND (char < = 39H)) THEN
            DO;
49    3               CALL co(char);
50    3               DO CASE count1;
51    4                    hours = SHL(hours,4) + (char - 30H);   /* input ASCII and convert to BCD */
52    4                    minutes = SHL(minutes,4) + (char - 30H);
53    4                    seconds = SHL(seconds,4) + (char - 30H);
```

280249-17

Figure 20 (Continued)

```
54   4            END;
55   3          END;
56   2        ELSE IF (char = ':') THEN DO;
58   3            count1 = count1 + 1;
59   3            CALL co(char);
60   3            END;
61   2          ELSE IF (char = 0DH) THEN not$done = FALSE;
63   2            ELSE CALL co(bell);
64   2        END;
65   1    CALL print$msg1;
66   1    hours = (SHR(hours,4) * 10) + (hours AND 0FH);      /* convert BCD to hex */
67   1    minutes = (SHR(minutes,4) * 10) + (minutes AND 0FH);
68   1    seconds = (SHR(seconds,4) * 10) + (seconds AND 0FH);
69   1    CALL print$msg1;
70   1    HSO_CMD = 38H;      /* set-up software-timer1 interrupt and TIMER1 as clock source */
71   1    tock = TIMER1 + 62500;                              /* load initial timer count for
                                                                interrupt */
72   1    HSO_TIME = tock;
73   1    INT_MASK = 20H;                                     /* set mask to select only software timer
                                                                interrupts */
74   1    INT_PENDING = 0;                                    /* clear interrupt pending register */
75   1    ENABLE;                                             /* enable interrupts */
76   1    DO FOREVER;                                         /* start the 'clock' */
77   2        IF tick THEN DO;
79   3          tick = FALSE;
80   3          seconds = seconds + 1;
          $CODE
81   3          IF (seconds = 60) THEN DO;
83   4            seconds = 0;
84   4            minutes = minutes + 1;
85   4            IF (minutes = 60) THEN DO;
87   5              minutes = 0;
88   5              hours = hours + 1;
89   5              IF (hours = 24) THEN hours = 0;
91   5              END;
92   4            END;
93   3          CALL ascii(seconds,.msg2(0));                 /* convert hex times to decimal
                                                                ASCII */
94   3          CALL ascii(minutes,.msg2(3));
95   3          CALL ascii(hours,.msg2(6));
96   3          IF not$first THEN DO;
98   4            DO I = 1 TO 8;                              /* backspace to beginning of line */
99   5              CALL co(BS);
100  5              END;
101  4            END;
102  3          DO I = 1 TO LENGTH(msg2);                     /* print the 'clock' time */
103  4            CALL co(msg2(I));
```

280249-18

Figure 20 (Continued)

```
104   4           END;
            $NOCODE
105   3         not$first = TRUE;
106   3         END;
107   2       END;
108   1   END   clock;

PL/M-96 COMPILER          iSBE-96 Sample Debug Program
                      ASSEMBLY LISTING OF OBJECT CODE

                                                ;    STATEMENT                81
01E7    993C0C              R                   CMPB    SECONDS,#3CH
01EA    D714                                    BNE     @0019
                                                ;    STATEMENT                83
01EC    110C                R                   CLRB    SECONDS
                                                ;    STATEMENT                84
01EE    170D                R                   INCB    MINUTES
                                                ;    STATEMENT                85
01F0    993C0D              R                   CMPB    MINUTES,#3CH
01F3    D70B                                    BNE     @0019
                                                ;    STATEMENT                87
01F5    110D                R                   CLRB    MINUTES
                                                ;    STATEMENT                88
01F7    170E                R                   INCB    HOURS
                                                ;    STATEMENT                89
01F9    99180E              R                   CMPB HOURS,#18H
01FC    D702                                    BNE     @0019
                                                ; STATEMENT                  90
01FE    110E                R                   CLRB    HOURS
                                                ; STATEMENT                  93
0200                            @0019:
0200    AC0C1C              R                   LDBZE TMP0,SECONDS
0203    C81C                                    PUSH    TMP0
0205    C90000              R                   PUSH    #MSG2
0208    2E60                                    CALL    ASCII
                                                ;    STATEMENT                94
020A    AC0D1C              R                   LDBZE TMP0,MINUTES
020D    C81C                                    PUSH    TMP0
020F    C90300              R                   PUSH    #MSG2+3H
0212    2E56                                    CALL    ASCII
                                                ;    STATEMENT                95
0214    AC0E1C              R                   LDBZE TMP0,HOURS
0217    C81C                                    PUSH    TMP0
0219    C90600              R                   PUSH    #MSG2+6H
021C    2E4C                                    CALL    ASCII
```

280249-19

Figure 20 (Continued)

```
                                                ;    STATEMENT                96
021E    301211          R                       BBC  NOTFIRST,0H,@001C
                                                ;    STATEMENT                99
0221    B1010A          R                       LDB  I,#1H
0224                        @001D:
0224    99080A          R                       CMPB I,#8H
0227    D909                                    BH   @001C
0229    C90800                                  PUSH #8H
022C    2E0B                                    CALL CO
                                                ;    STATEMENT               100
022E    170A            R                       INCB I
0230    D7F2                                    BNE  @001D
                                                ;    STATEMENT               102
0232                        @001C:
                                                ;    STATEMENT               103
0232    B1010A          R                       LDB  I,#1H
0235                        @001F:
0235    AC0A1C          R                       LDBZE TMP0,I
0238    8908001C                                CMP   TMP0,#8H
023C    D910                                    BH    @0020
023E    AC0A1C          R                       LDBZE TMP0,I
0241    AF1D00001C      R                       LDBZE TMP0,MSG2[TMP0]
0246    C81C                                    PUSH  TMP0
0248    2DEF                                    CALL  CO
                                                ;    STATEMENT               104
024A    170A            R                       INCB I
024C    D7E7                                    BNE  @001F
024E                        @0020:

MODULE INFORMATION:
    CODE AREA SIZE              = 0231H     561D
    CONSTANT AREA SIZE          = 0022H      34D
    DATA AREA SIZE              = 0000H       0D
    STATIC REGS AREA SIZE       = 0019H      25D
    OVERLAYABLE REGS AREA SIZE  = 0000H       0D
    MAXIMUM STACK SIZE          = 000AH      10D
    145 LINES READ
PL/M-96 COMPILATION COMPLETE.   0 WARNINGS,     0 ERRORS
```

280249-20

Figure 20 (Continued)

```
MCS-96 MACRO ASSEMBLER     Sample Debug Program -Interrupt Service Routine
DOS MCS-96 MACRO ASSEMBLER, V1.0
SOURCE FILE: TIMER.A96
OBJECT FILE: TIMER.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: <none>
ERR  LOC   OBJECT          LINE        SOURCE STATEMENT
                             1     $TITLE ('Sample Debug Program -Interrupt Service
                                   Routine')
                             2
                             3
                             4     TIMER     MODULE
                             5
                             6     ;Externals
                             7
                             8     EXTRN          tick      :BYTE ;tick is declared FAST
                                                              so will be in internal RAM
                             9     EXTRN          tock      :WORD ;contains first
                                                              HSO_TIME setting
                            10
                            11     ;Publics
                            12
                            13     PUBLIC         count
                            14
                            15     ;Local variables
                            16
      0004                  17     HSO_TIME       EQU       04H:WORD ; Write only
      0006                  18     HSO_CMD        EQU       06H:BYTE ; Write only
      000A                  19     TIMER1         EQU       0AH:WORD ; Read only
                            20
    0000                    21     RSEG
                            22
    0000                    23     count:         DSB       1
                            24
                            25     ;vector table - only the software timer should be accessed
                            26
    2000                    27     CSEG at 2000h
                            28
    2000   1800       R     29         DCW        oops      ;timer_Overflow
    2002   1800       R     30         DCW        oops      ;ADdone
    2004   1800       R     31         DCW        oops       ;HSI_Data_Available
    2006   1800       R     32         DCW        oops      ;HSO_Execution
    2008   1800       R     33         DCW        oops      ;HSI0
    200A   0000       R     34         DCW        tovfl     ;SW_timers
    200C   1800       R     35         DCW        oops      ;Serial_IO
    200E   1800       R     36         DCW        oops      ;External_Interrupt
                            37
                            38     ;service routines
```

280249-21

Figure 21

```
                                39
0000                            40      CSEG
                                41
0000    F2                      42      Tovfl:      PUSHF
0001    1700            R       43                  INCB    count
0003    990800          R       44                  CMPB    count,#8
0006    D706                    45                  JNE     loop1
0008    B1FF00          E       46                  LDB     tick,#0FFH ;set 'tick' = TRUE
000B    B10000          R       47                  LDB     count,#0
000E    B13806                  48      loop1:      LDB     HSO_CMD,#38H ;reload HSO
                                                                CAM
0011    4524F40004      E       49                  ADD     HSO_TIME,tock,#62500
0016    F3                      50                  POPF
0017    F0                      51                  RET
                                52
0018    F2                      53      Oops:       PUSHF   ;arriving here means an
                                                                interrupt occurred which
0019    FD                      54                  NOP     ; should not have occurred.
                                                                This is also used to
001A    FD                      55                  NOP     ; initialize all the interrupt
                                                                vectors for bebugging
001B    F3                      56                  POPF    ; with the iSBE-96.
001C    F0                      57                  RET
                                58
001D                            59      END

MCS-96 MACRO ASSEMBLER    Sample Debug Program -Interrupt Service Routine

SYMBOL TABLE LISTING
--------------------

            NAME                    VALUE     ATTRIBUTES
COUNT . . . . . .                   0000H     REG  REL  PUBLIC  BYTE
HSO_CMD . . . . . .                 0006H     NULL  ABS  BYTE
HSO_TIME . . . . . .                0004H     NULL  ABS  WORD
LOOP1 . . . . . .                   000EH     CODE  REL  ENTRY
OOPS . . . . . .                    0018H     CODE  REL  ENTRY
TICK . . . . . .                    -----     NULL  EXTERNAL  BYTE
TIMER . . . . . .                   -----     MODULE  STACKSIZE(0)
TIMER1 . . . . . .                  000AH     NULL  ABS  WORD
TOCK . . . . . .                    -----     NULL  EXTERNAL  WORD
TOVFL . . . . . .                   0000H     CODE  REL  ENTRY

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

280249-22

Figure 21 (Continued)

Since it is much easier to have an integer number for a loop counter, by setting the number of time-outs to eight we find that the count needed is 62,500. This number may eventually have to be tweaked because we did not account for the time required to service the interrupt itself or the tolerance of the 12 MHz crystal on the iSBE-96 emulator, but for our purposes it is close enough.

After prompting for the initial time, the sample program converts the input ASCII characters to hexadecimal. It then initializes software timer 1 to use TIMER 1 as a clock source and signal for an interrupt upon reaching the specified time (a count of 62,500), which is then input to the HSO time register. The software timer interrupt service routine keeps count of the number of times it is activated and on the eighth pass it sets a flag which allows the mainline program to increment the 'clock'. The current 'clock' time is then converted to decimal ASCII and displayed on the terminal connected to the iSBE-96 DTE serial port.

Sample Debug Session

After generating the files CLOCK.P96 and TIMER.A96 as shown in Figures 20 and 21 respectively, use the DOS batch file as show to generate the absolutely located object code (CLOCK.ABS). Figure 20 contains a partial assembly code listing of the PL/M-96 program module (compiled with the CODE and NOCODE controls). The code listing is needed for debugging with the iSBE-96 emulator since it does not support PL/M-96 symbols or line numbers. For the sake of a manageable illustration only part of the assembly code was generated for the PL/M-96 module. The segment map and symbol table generated by RL96 for the sample program (CLOCK.M96) is shown in Figure 22. The segment map shows the address of the instructions of the program since the addresses of the relocatable code in the listing are only relative module addresses.

Once the linked object module has been generated, invoke the iSBE-96 driver software which will sign on with the version number and establish communications with the iSBE-96 emulator. The sample program can then be loaded by typing LOAD CLOCK.ABS <cr>. After the sample program object code has been loaded, begin emulation by typing GO.

You will now be prompted on the terminal to set the current time, 'set time hh:mm:ss <cr>' where <cr> represents a carriage return or enter. After entering the time and carriage return, you will notice that the 'clock' display appears to backup across the screen on the terminal. If you look closely, the hours and seconds also appear to be transposed. Press the escape key on the IBM PC XT or AT, (referred to from now on as the console) to stop emulation. It should be clear that our sample program has two separate problems, relative clock print-out position and transposed hours and seconds.

First let's tackle the print-out position problem. By referring to the PL/M-96 module listing (Figure 20), we discover that the current time is printed out by the DO loop in lines 102 through 104. If you compare these lines with procedure 'print$msg1', you will see that the message index in line 103 should be I-1. This would cause us to only print out 7 of the eight characters. But, the DO loop in lines 98 through 100 backspaces eight characters. These could very well cause the position problem.

To confirm this we first need to consult the assembly code listing section of the PL/M-96 module listing and the link map (Figures 20 and 23), to obtain the address of line 102. The associated line number is printed on the right-hand margin in the assembly code section of the PL/M-96 listings (Figure 20). Since PL/M-96 always places procedures and constants at the beginning of code, the start address for line 102 is 0232H + 2084H = 22B6H. To verify this we can type DASM 22B6 to 22D5 on the console. The resultant dissassembly display is shown in Figure 24. After comparing the display to the listing we can verify that we have the correct address.

To correct the problem we need to load TMP0 (1CH) with I-1 (2EH) and, because TMP0 is then used as an index, we need to ensure that the high byte (1DH) for word pointer 1CH is clear. As you probably already have guessed, the three byte instruction at 22C2H does not give us enough room to do all that. Therefore, we must branch to a non-used area (above 230DH from the link map), add the necessary instructions, and then branch back into the instruction stream. This can be done by typing the following on the console:

```
ASM 22C2 = BR +4AH <cr>,<cr>
ASM 230E = LDB 1C,2E <cr>
DECB 1C <cr>
CLRB 1D <cr>
BR -53H <cr>,<cr>
```

```
plm96 clock.p96
asm96 timer.a96
rl96 clock.obj,timer.obj,plm96.lib to clock.abs stacksize(20H)
```

Figure 22

```
DOS MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: CLOCK.OBJ, TIMER.OBJ, PLM96.LIB
OUTPUT FILE: CLOCK.ABS
CONTROLS SPECIFIED IN INVOCATION COMMAND:
    STACKSIZE(20H)


INPUT MODULES INCLUDED:
    CLOCK.OBJ(CLOCK)     01/14/86     13:28:27
    TIMER.OBJ(TIMER)     01/14/86     13:28:38
    PLM96.LIB(PLMREG)    11/02/83
    PLM96.LIB(TIME)      11/02/83

SEGMENT MAP FOR CLOCK.ABS(CLOCK):

                    TYPE    BASE      LENGTH    ALIGNMENT     MODULE NAME
                    ----    ----      -------   ----------    -------------

**RESERVED*                 0000H     001AH
                    REG     001AH     0001H     BYTE          TIMER
*** GAP ***                 001BH     0001H
                    REG     001CH     0008H     ABSOLUTE      PLMREG
                    REG     0024H     0019H     WORD          CLOCK
*** GAP ***                 003DH     0001H
                    STACK   003EH     0020H     WORD
*** GAP ***                 005EH     1F86H
                    DATA    1FE4H     0002H     ABSOLUTE      CLOCK
                    DATA    1FE6H     0002H     ABSOLUTE      CLOCK
*** GAP ***                 1FE8H     0018H
                    CODE    2000H     0010H     ABSOLUTE      TIMER
*** GAP ***                 2010H     0070H
                    CODE    2080H     0003H     ABSOLUTE      CLOCK
*** GAP ***                 2083H     0001H
                    CODE    2084H     0253H     WORD          CLOCK
                    CODE    22D7H     001DH     BYTE          TIMER
                    CODE    22F4H     0019H     BYTE          TIME
***GAP ***                  230DH     DCF3H

ATTRIBUTES          VALUE     NAME

SYMBOL TABLE FOR CLOCK.ABS(CLOCK):

                              PUBLICS:
REG       BYTE      0033H       TICK
REG       WORD      002CH       TOCK
CODE      ENTRY     20A6H       CI
```

280249-23

Figure 23

```
CODE    ENTRY   20BDH   CO
CODE    ENTRY   20DAH   INITDT
CODE    ENTRY   20EEH   ASCII
REG     BYTE    001AH   COUNT
REG     NULL    001CH   PLMREG
CODE    ENTRY   22F4H   ??TIME
NULL    NULL    005EH   MEMORY
NULL    NULL    1F86H   ?MEMORY_SIZE

                        MODULE: CLOCK

                        MODULE: TIMER

                        MODULE: PLMREG

                        MODULE: TIME

RL 96 COMPLETED,     0 WARNING(S),     0 ERROR(S)
```

Figure 23 (Continued)

```
*dasm 22b6 to 22d5

ADDRESS   DATA          MNEMONIC   OPERANDS
22B6H     B1012E          LDB      2E,#01
22B9H     AC2E1C          LDBZE    1C,2E
22BCH     8908001C        CMP      1C,#0008
22C0H     D910            JH       $+12
22C2H     AC2E1C          LDBZE    1C,2E
22C5H     AF1D24001C      LDBZE    1C,0024[1C]
22CAH     C81C            PUSH     1C
22CCH     2DEF            SCALL    $-020F
22CEH     172E            INCB     2E
22D0H     D7E7            JNE      $-17
22D2H     B1FF36          LDB      36,#FF

*
```

Figure 24

We must now restart emulation to see if this patch fixes the position problem. To restart emulation type GO FROM 2080 on the console. After setting the time on the terminal, we see that this did fix the position problem.

Now to fix the problem with the hours and seconds transposed on the 'clock' print-out. By consulting the PL/M-96 module listing (Figure 20), we see that the times are converted and put into printable message format by lines 93 through 95. Comparing those lines with the format declarations of messages 2a and 3 in line 2, we see that lines 93 and 95 use the wrong index into message 2 for storing seconds and hours.

To confirm this we again need to consult the assembly code listing section of the PL/M-96 module listing and the link map (Figures 20 and 23), to obtain the address of line 93. The address for line 93 turns out to be 0200H + 2084H = 2284H. We verify this by typing DASM 2284 TO 22AA on the console. After comparing the resultant display (Figure 25) and the code listing, we can see that we have the correct address. To correct the problem we need to swap the instruction at 2289H with the instruction at 229DH. This can be done by typing the following on the console:

```
ASM 2298 = PUSH #2A <cr>,<cr>
ASM 2290 = PUSH #24 <cr>,<cr>
```

We must now restart emulation to see if this fixes the problem. To restart emulation where we left off, type GO on the console. Checking the terminal, we can see that this does fix the transposition problem and the 'clock' print-out is correct.

Now that we have confirmed that our fixes correct the problems, the PL/M-96 module should be updated to incorporate those corrections. The debugged PL/M-96 module is illustrated in Figure 26.

```
*dasm 2284 to 22aa

ADDRESS   DATA     MNEMONIC   OPERANDS
2284H     AC301C     LDBZE    1C,30
2287H     C81C       PUSH     1C
2289H     C92400     PUSH     #0024
228CH     2E60       SCALL    $-019E
228EH     AC311C     LDBZE    1C,31
2291H     C81C       PUSH     1C
2293H     C92700     PUSH     #0027
2296H     2E56       SCALL    $-01A8
2298H     AC321C     LDBZE    1C,32
229BH     C81C       PUSH     1C
229DH     C92A00     PUSH     #002A
22A0H     2E4C       SCALL    $-01B2
22A2H     303611     JBC      36,00,$+14
22A5H     B1012E     LDB      2E,#01
22A8H     99082E     CMPB     2E,#08

*
```

Figure 25

```
$title (' iSBE-96 Sample Debug Program')
$optimize (3)
clock: DO;
/* local declarations */
DECLARE     bell                          LITERALLY   '07H',
            BS                            LITERALLY   '08H',
            FOREVER                       LITERALLY   'WHILE 1',
            FALSE                         LITERALLY   '0',
            TRUE                          LITERALLY   'NOT FALSE',
            BOOLEAN                       LITERALLY   'BYTE',
            msg1(*)                       BYTE        DATA(0dH,0aH),
            msg2a(*)                      BYTE        DATA(0,0,':',0,0,':',0,0),
            msg2(8)                       BYTE        FAST,
            msg3(*)                       BYTE        DATA('set time - hh:mm:ss <cr>'),
            (I,char)                      BYTE,
            seconds                       BYTE,
            minutes                       BYTE,
            hours                         BYTE,
            tick                          BYTE        FAST        PUBLIC,
            tockWORDPUBLIC,
            count                         BYTE        EXTERNAL,
            count1                        BYTE,
            not$done                      BOOLEAN,
            not$first                     BOOLEAN,
            HSO_TIME                      WORD        AT (04H),
            HSO_CMD                       BYTE        AT (06H),
            INT_MASK                      BYTE        AT (08H),
            INT_PENDINGBYTEAT (09H),
            TIMER1                        WORD        AT (0AH),
            dt_data                       ADDRESS     AT (1FE4H),
            dt_status                     ADDRESS     AT (1FE6H);
/* Procedure declarations */
ci:    PROCEDURE      BYTE      PUBLIC;
DO WHILE ((dt_status AND 02H) = 0H);              /* wait till RxRDY */
    END;
char = dt_data AND 7FH;
RETURN char;
END  ci;
co:    PROCEDURE      (char)    PUBLIC;
DECLARE    char    BYTE;
DO WHILE ((dt_status AND 1) = 0 );                /* wait till TxRDY */
    END;
dt_data = char;
END    co;
init$DT:    PROCEDURE      PUBLIC;
dt_status = 37H;                                  /* clear any errors on the DTs 8251A USART */
```

280249-24

Figure 26

```
CALL TIME(1);
char = dt_data;                                    /* clear the DT receive buffer */
END     init$DT;
ascii:     PROCEDURE (value,dest$ptr)      PUBLIC;
DECLARE (value,temp)       BYTE,
          dest$ptr       ADDRESS,
          (dest    BASED    dest$ptr)  (2)    BYTE;
value = SHL((value/10),4) + (value MOD 10);        /* convert to BCD */
temp = value;
dest(0) = SHR(temp,4) + 30H;                       /* convert to ASCII decimal value */
dest(1) = (value AND 0FH) + 30H;
END ascii;
print$msg1:     PROCEDURE;
DECLARE     I     BYTE;
DO I = 1 TO LENGTH(msg1);
    CALL co(msg1(I-1));
    END;
END     print$msg1;
/* Program starts here */
CALL TIME(50);                                     /* delay to insure iSBE-96 NMI line is stable */
CALL init$DT;                                      /* initialize DT serial port */
count,count1 = 0;                                  /* initialize variables */
not$done = TRUE;
not$first,tick = FALSE;
seconds,minutes,hours = 0;
CALL movb(.msg2a,.msg2,LENGTH(msg2a));
CALL print$msg1;
DO I = 1 TO LENGTH(msg3);                          /* query for initial time */
    CALL co(msg3(I-1));
    END;
CALL print$msg1;
DO WHILE not$done;                                 /* input initial time values */
    char = ci;
    IF ((char > =30H) AND (char < =39H)) THEN DO;
      CALL co(char);
      DO CASE count1;
        hours = SHL(hours,4) + (char - 30H);       /* input ASCII and convert to BCD */
        minutes = SHL(minutes,4) + (char - 30H);
        seconds = SHL(seconds,4) + (char - 30H);
        END;
      END;
    ELSE IF (char = ':') THEN DO;
        count1 = count1 + 1;
        CALL co(char);
        END;
      ELSE IF (char = 0DH) THEN not$done = FALSE;
```

280249-25

Figure 26 (Continued)

```
        ELSE CALL co(bell);
    END;
CALL print$msg1;
hours = (SHR(hours,4) * 10) + (hours AND 0FH);          /* convert BCD to hex */
minutes = (SHR(minutes,4) * 10) + (minutes AND 0FH);
seconds = (SHR(seconds,4) * 10) + (seconds AND 0FH);
CALL print$msg1;
HSO_CMD = 38H;                                          /* set-up software-timer1 interrupt and
                                                           TIMER1 as clock source */
tock = TIMER1 + 62500;                                  /* load initial timer count for interrupt */
HSO_TIME = tock;
INT_MASK = 20H;                                         /* set mask to select only software timer
                                                           interrupts */
INT_PENDING = 0;                                        /* clear interrupt pending register */
ENABLE;                                                 /* enable interrupts */
DO FOREVER;                                             /* start the 'clock' */
    IF tick THEN DO;
      tick = FALSE;
      seconds = seconds + 1;
$CODE
      IF (seconds = 60) THEN DO;
        seconds = 0;
        minutes = minutes + 1;
        IF (minutes = 60) THEN DO;
          minutes = 0;
          hours = hours + 1;
          IF (hours = 24) THEN hours = 0;
          END;
        END;
CALL ascii(seconds,.msg2(6));                           /* convert hex times to decimal ASCII */
CALL ascii(minutes,.msg2(3));
CALL ascii(hours,.msg2(0));
      IF not$first THEN DO;
        DO I = 1 TO 8;                                  /* backspace to beginning of line */
          CALL co(BS);
          END;
        END;
      DO I = 1 TO LENGTH(msg2);                         /* print the 'clock' time */
        CALL co(msg2(I-1));
        END;
$NOCODE
      not$first = TRUE;
      END;
    END;
END clock;
```

280249-26

Figure 26 (Continued)